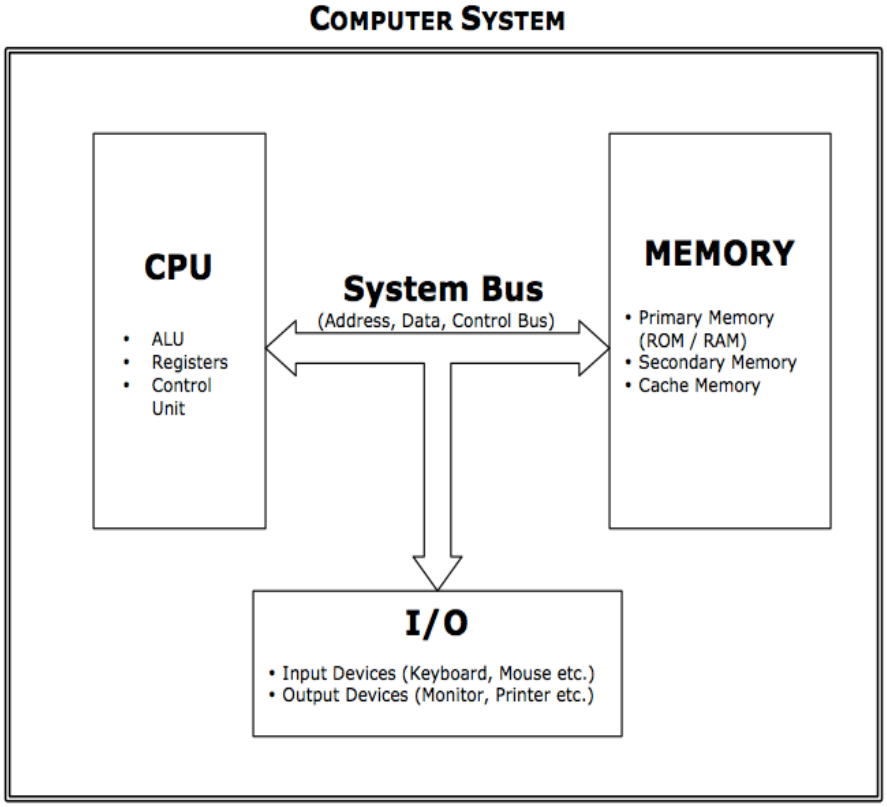
BASIC ORGANIZATION OF A COMPUTER



A computer system mainly consists of The CPU, Memory, I/O devices and the System bus used for interconnections

CPU

1) The Central Processing Unit is the **most important part** of the Computer.
2) It is also called the **microprocessor** or simply the **processor**.
3) It consists of the ALU, Registers, Control Unit etc.
4) All **programs are executed** in the CPU.
5) A program is a **set of instructions** stored in the memory.
6) The main function of the CPU is to **fetch, decode and execute** these instructions.
7) Instructions are **fetched from the memory** using the various **buses**.
8) Thereafter they are **decoded by the Control Unit** to analyze the Opcode.
9) Finally the instruction is **executed** to perform the **desired operation**.
10) This **execution** mainly involves the **ALU** and the **internal registers** of the processor.

MEMORY

1) The memory is used to **store information**.
2) It mainly stores **programs and data**.
3) Memory has various **locations**.
4) Each location is identified by its own **unique address and contains some data**.
5) The most basic form of memory is called **Primary Memory**, which consists of **RAM and ROM**.
6) Then there are **secondary** storage devices such as **hard disk**.
7) There are **portable** storage devices like **CD, DVD, Pen drives** etc.
8) Finally, there is a **high-speed memory called Cache** memory implemented using **SRAM**.
9) **RAM is "Volatile"**, that means contents of RAM are **lost** after power supply is withdrawn.
10) All **other** memories are **Non-Volatile** memories.

I/O

1) I/O devices are used for the **flow of information in and out** of the computer system.
2) **Input** devices such as **keyboard, mouse**, etc. are used to provide inputs into the computer.
3) They are used to **enter programs and data**.
4) **Output** devices such as **monitor and printer** are used to **generate results**.
5) Some devices such as a **touch-screen** can be used for **both input and output**.

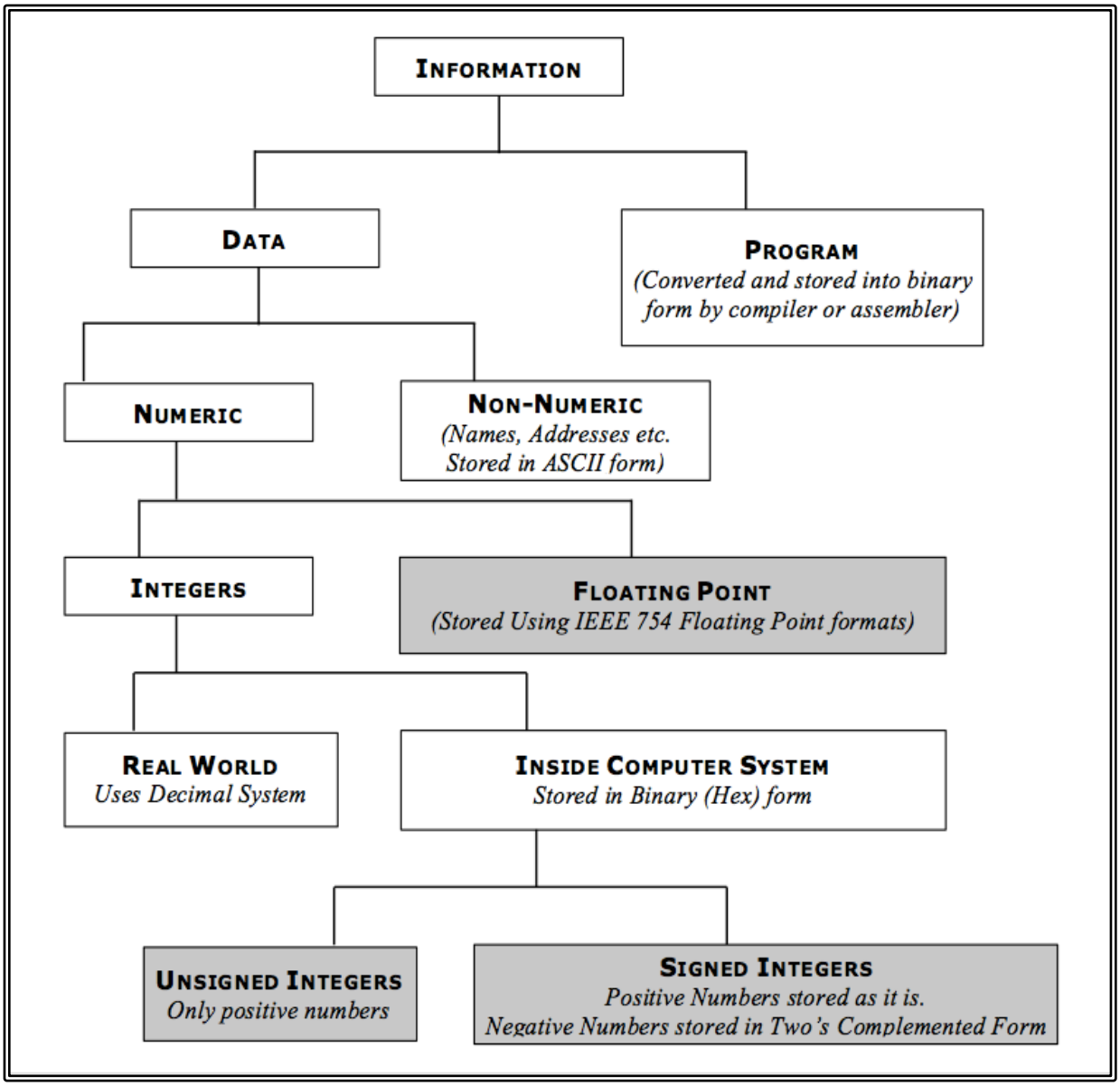
System Bus

1) A bus is a set of **interconnecting lines used to carry information**.
2) **Size** of a bus means its **number of lines**.
3) An 8-bit bus has eight lines carrying **one bit each**.
4) There are three types of buses.
5) **Address Bus: It carries the address for the operation.**
   During any operation, the address bus identifies the location where the operation is performed.
   The size of the address bus determines the amount of Primary Memory that can be connected.
   Example: If address bus is 16-bit, we can connect 2^{16} = 64 KB Memory.
   Bigger the address bus, bigger is the memory.
6) **Data Bus: It carries data to and from the processor.**
   The size of data bus determines how much data can be transferred in one operation (cycle).
   Bigger the data bus, faster the processor, as it can transfer more data in one cycle.
7) **Control Bus: It Carries control signals like RD, WR etc.**
   These signals determine the kind of operation that will be performed on the system bus.

DATA REPRESENTATION & ARITHMETIC ALGORITHMS

INFORMATION

Information stored in the memory can be of the following categories.



UNSIGNED INTEGERS

These are binary numbers that are always assumed to be positive.
Here all available bits of the number are used to represent the magnitude of the number.
No bits are used to indicate its sign, hence they are called unsigned numbers.
E.g.: Roll Numbers, Memory addresses etc.

SIGNED INTEGERS

These are binary numbers that can be either positive or negative.
The MSB of the number indicates whether it is positive or negative.
If MSB is 0 then the number is Positive.
If MSB is 1 then the number is Negative.
Negative numbers are always stored in their 2's Complement form.

NUMBER	SIGN MAGNITUDE	ONE'S COMPLEMENT	TWO'S COMPLEMENT
3	011	011	011
2	010	010	010
1	001	001	001
0	000	000	000
-0	100	111	000
-1	101	110	011
-2	110	101	110
-3	111	100	101
-4	---	---	100

Negative numbers are stored in 2's Complement form due to two reasons as highlighted above.

1) **Two's complement gives a unique representation for zero.**
   Any other system gives a separate representation for +0 and for -0.
   This is absurd.
   In two's complement system, -(x) is stored as two's complement of (x).
   Applying the same rule for 0, -(0) should be stored as two's complement of 0.
   0 is stored as 000.
   So –(0) should be stored as two's complement of 000, which again is 000.
   Hence two's complement gives a unique representation for 0.

2) **It produces an additional number on the negative side.**
   As two's complement system produces a unique combination for 0, it has a spare combination '100' in the above case, and can be used to represent –(4).

INTEGER NUMBER REPRESENTATION

3 BIT INTEGER	
2^3 = 8 therefore 8 combinations	
Unsigned	Signed
0 ... 7	-4 ... -1 0 1 ... 3

4 BIT INTEGER	
2^4 = 16 therefore 16 combinations	
Unsigned	Signed
0 ... 15	-8 ... -1 0 1 ... 7

5 BIT INTEGER	
2^5 = 32 therefore 32 combinations	
Unsigned	Signed
0 ... 31	-16 ... -1 0 1 ... 15

8 BIT INTEGER	
2^8 = 256 therefore 256 combinations	
Unsigned	Signed
0 ... 255	-128 ... -1 0 1 ... 127

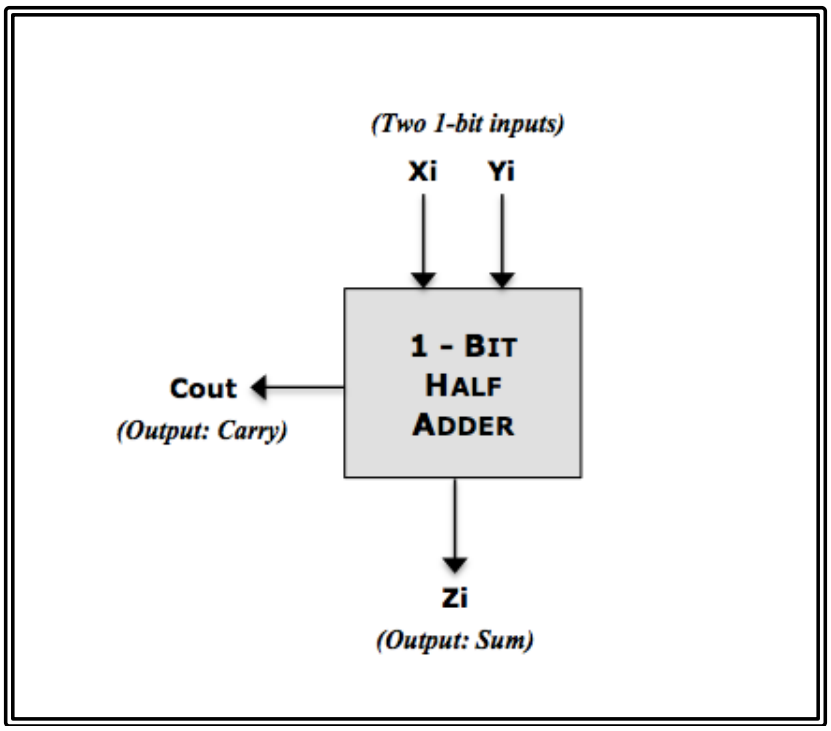
16 BIT INTEGER	
2^{16} = 65536 therefore 65536 combinations	
Unsigned	Signed
0 ... 65535	-32768 ... -1 0 1 ... 32767

SIGNED NUMBER EXAMPLES		
+5	0101	
-5	1011	← *2's complement*
+9	01001	
-9	10111	← *2's complement*
+23	010111	
-23	101001	← *2's complement*

INTEGER DATA COMPUTATION

ONE BIT ADDITION: HALF ADDER

1) It is a simple 1-bit adder circuit.
2) It adds two 1-bit inputs Xi & Yi and produces a sum Zi and a Carry Cout.
3) As it does not consider any carry input, it can't be combined to add large numbers.
4) Hence it is called a Half Adder.



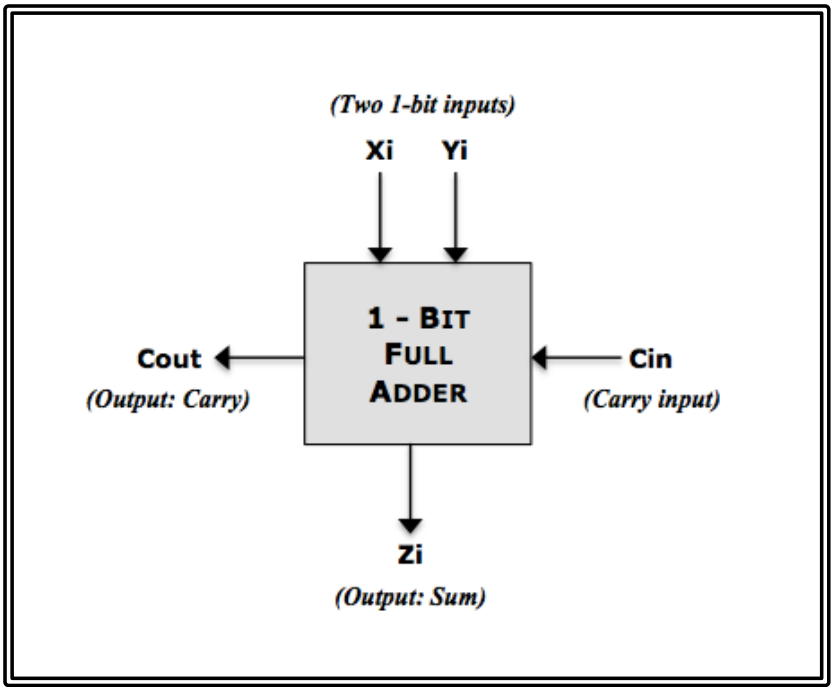
Inputs bits: Xi and Yi.

Output (Sum): Zi
Output (Carry): Cout

Formula:
Sum (Zi) = Xi Ex-Or Yi.
Carry (Cout) = Xi.Yi

ONE BIT ADDITION: FULL ADDER

1) It is a **1-bit adder** circuit.
2) It adds **two 1-bit inputs Xi & Yi**, along with a **Carry Input Cin**.
3) It produces a **sum Zi** and a Carry output **Cout**.
4) As it considers a carry input, it can be used in **combination to add large numbers**.
5) Hence it is called a **Full Adder**.



Inputs bits: Xi and Yi.
Input Carry: Cin

Output (Sum): Zi
Output (Carry): Cout

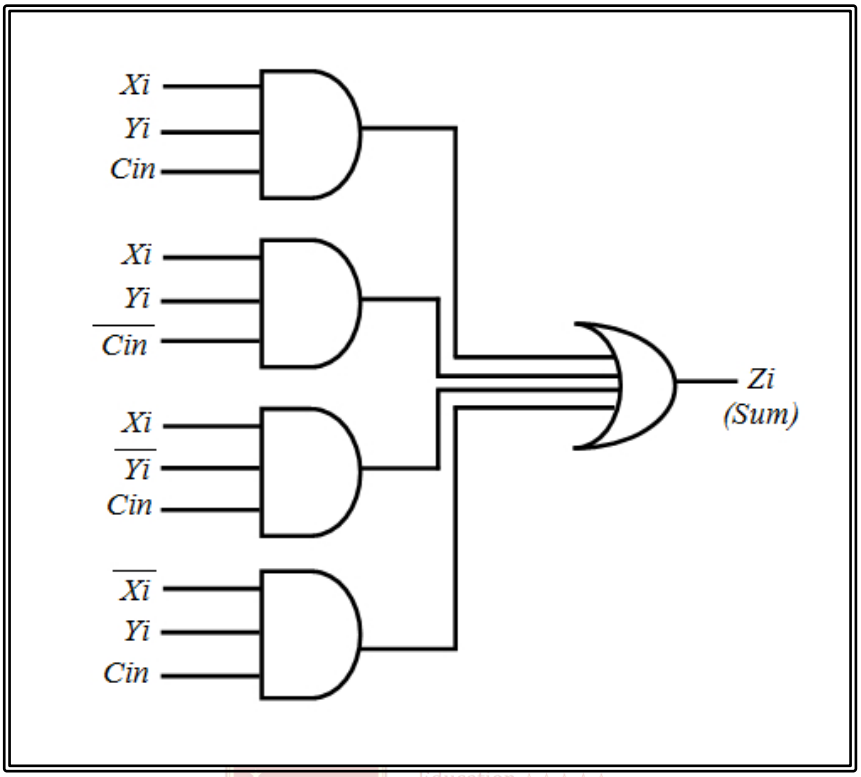
Formula for Sum (Zi)

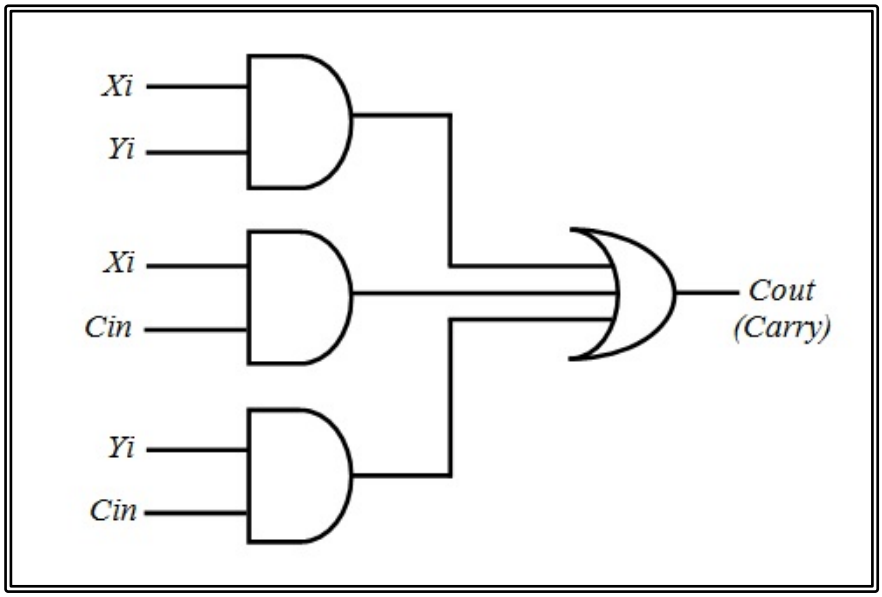
$$Zi = Xi \oplus Yi \oplus Cin$$
$$\therefore Zi = Xi \cdot Yi \cdot Cin + Xi \cdot Yi \cdot \overline{Cin} + Xi \cdot \overline{Yi} \cdot Cin + \overline{Xi} \cdot Yi \cdot Cin$$

Formula for Carry (Cout)

$$Cout = Xi \cdot Yi + Xi \cdot Cin + Yi \cdot Cin$$

CIRCUIT FOR A FULL ADDER





MULTIPLE BIT ADDITION: SERIAL ADDER / RIPPLE CARRY ADDER

1) A Full Adder can add two "1-bit" numbers with a Carry input.
2) It produces a "1-bit" Sum and a Carry output.
3) **Combining many of these Full Adders, we can add multiple bits**.
4) One such method is called Serial Adder.
5) Here, bits are **added one-by-one from LSB onwards**.
6) The **Carry of each stage is propagated (Rippled) into the next stage**.
7) Hence, these adders are also called **Ripple Carry Adders**.
8) Advantage: They are very **easy to construct**.
9) Drawback: As addition happens **bit-by-bit**, they are **slow**.
10) **Number of cycles** needed for the addition is equal to the **number of bits to be added**.

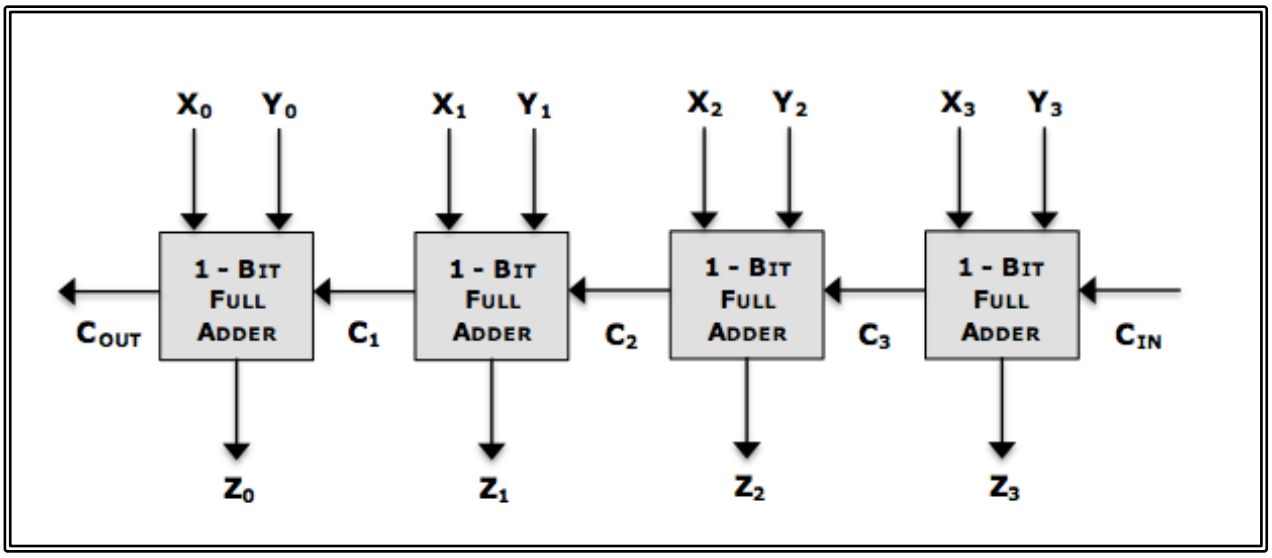
Inputs:
Assume X and Y are two "4-bit" numbers to be added, along with a Carry input C_{IN}.

$X = X_0\ X_1\ X_2\ X_3$ (X_0 is the MSB … X_3 is the LSB)
$Y = Y_0\ Y_1\ Y_2\ Y_3$ (Y_0 is the MSB … Y_3 is the LSB)
C_{IN} = Carry Input

Outputs:
Assume Z to be a "4-bit" output, and C_{OUT} to be the output Carry

$Z = Z_0\ Z_1\ Z_2\ Z_3$ (Z_0 is the MSB … Z_3 is the LSB)
C_{OUT} = Carry Output

Circuit for 4-bit Serial Adder/ Ripple Carry Adder



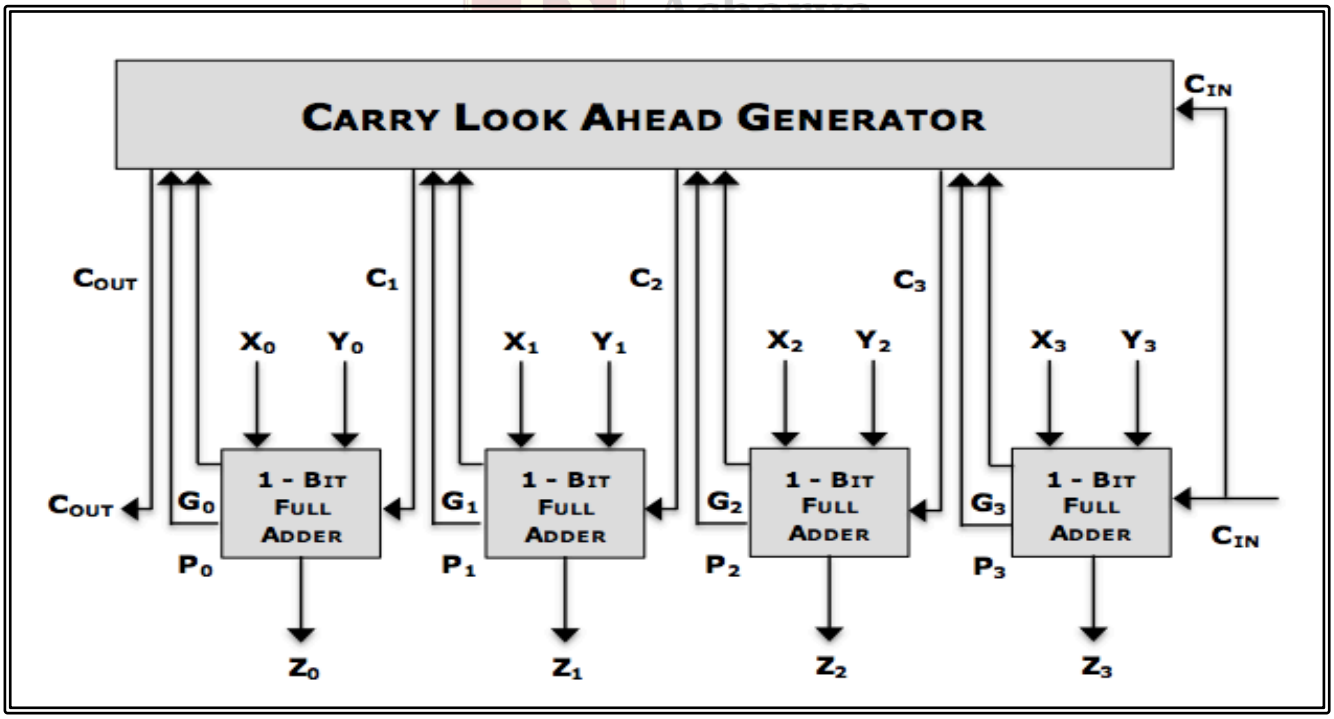
MULTIPLE BIT ADDITION: CARRY LOOK AHEAD ADDER / PARALLEL ADDER

1) It is used to add multiple bits **simultaneously**.
2) While adding multiple bits, the main issue is that of the **intermediate carries**.
3) In Serial Adders, we therefore added the bits one-by-one.
4) This allowed the carry at any stage to propagate to the next stage.
5) But this also made the process **very slow**.
6) If we "**PREDICT**" the **intermediate carries**, then all bits can be added **simultaneously**.
7) This is done by the **Carry Look Ahead Generator** Circuit.
8) Once all carries are determined beforehand, then all bits can be **added simultaneously**.
9) Advantage: This makes the addition process **extremely fast**.
10) Drawback: Circuit is **complex**.

Inputs:
Assume X and Y are two "4-bit" numbers to be added, along with a Carry input C_{IN}.
$X = X_0\ X_1\ X_2\ X_3$ (X_0 is the MSB ... X_3 is the LSB); $Y = Y_0\ Y_1\ Y_2\ Y_3$ & C_{IN} = Carry Input

Outputs:
Assume Z to be a "4-bit" output, and C_{OUT} to be the output Carry
$Z = Z_0\ Z_1\ Z_2\ Z_3$ & C_{OUT} = Carry Output

Circuit for 4-bit Serial Adder/ Ripple Carry Adder



CALCULATIONS

We can "Predict" (Look Ahead) all the intermediate carries in the following manner.

The Carry at any stage can be calculated as:

$C_i = X_i.Y_i + X_i.C_{in} + Y_i.C_{in}$
$C_i = X_i.Y_i + C_{in}(X_i + Y_i)$

$$\mathbf{C_i = G_i + P_i.C_{IN}}$$

Here $Gi = X_i.Y_i$ … (Generate)
And $Pi = X_i+Y_i$ … (Propagate)

We need to predict the Carries: C_3, C_2, C_1 and C_0

$C_3 = G_3 + P_3C_{IN}$	… I

$C_2 = G_2 + P_2C_3$

Substituting the value of C3, we get:

$C_2 = G_2 + P_2G_3 + P_2P_3C_{IN}$	… II

$C_1 = G_1 + P_1C_2$

Substituting the value of C2, we get:

$C_1 = G_1 + P_1G_2 + P_1P_2G_3 + P_1P_2P_3C_{IN}$	… III

$C_0 = G_0 + P_0C_1$

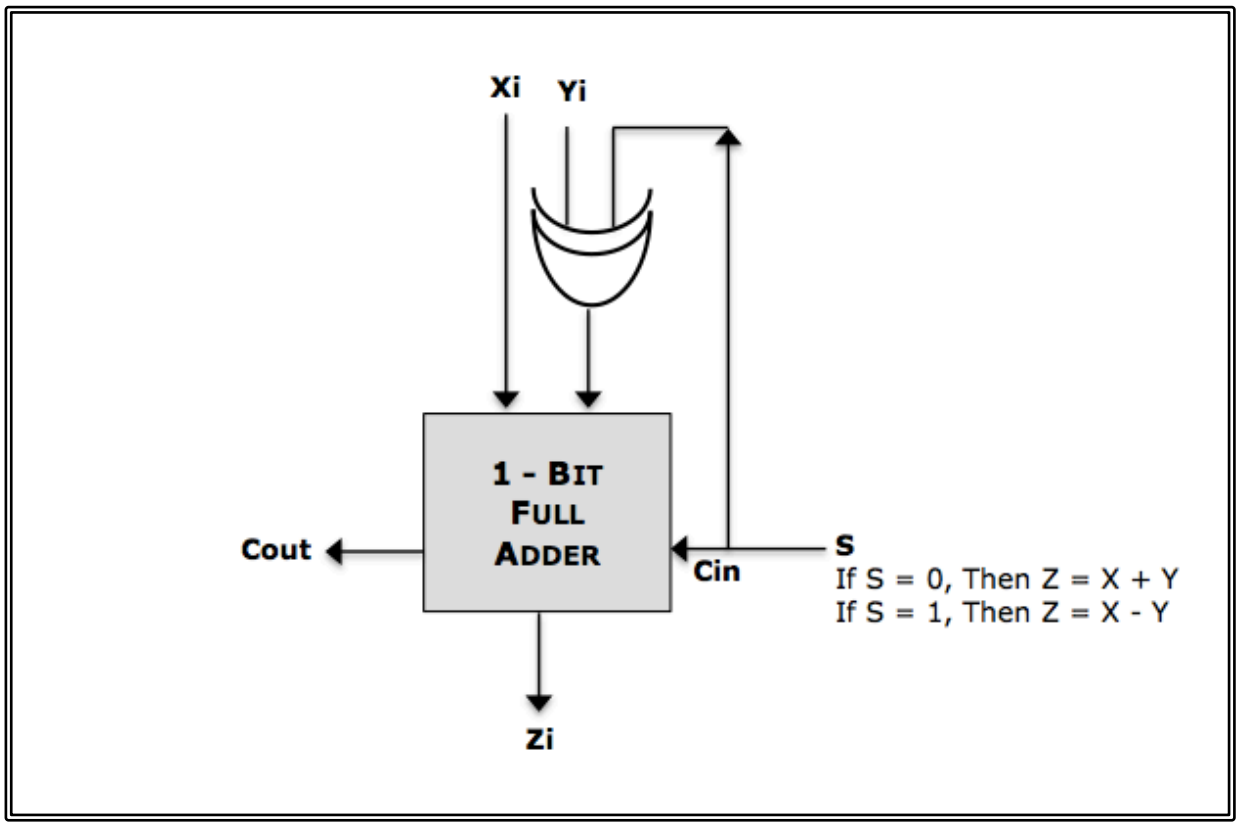
Substituting the value of C1, we get:

$C_0 = G_0 + P_0G_1 + P_0P_1G_2 + P_0P_1P_2G_3 + P_0P_1P_2P_3C_{IN}$	… IV

From the above four equations, it is clear that the values of all the four Carries (C_3, C_2, C_1, C_0) can be determined beforehand even without doing the respective additions. To do this we need the values of all G's ($X_i.Y_i$) and all P's (X_i+Y_i) and the original carry input C_{IN}. This is done by the Carry Look Ahead Generator Circuit.

ADDER / SUBTRACTOR CIRCUIT:

1) **Subtraction** in binary numbers is simply performed by **addition of two's complement**.
2) That means, a special circuit for subtraction is not needed.
3) The same circuit that is used for Addition, can also be used for subtraction.
4) The following circuit is called **Adder/ Subtractor** circuit.
5) It can perform Addition as Z = X + Y.
6) It can also perform subtraction as Z = X + (2's Complement of Y)
7) The Variable "S" determines if Addition or Subtraction will be performed.
8) **If S = 0, then Addition will be performed**.
9) **If S = 1, then Subtraction will be performed**.
10) If S = 1, then the operation is **Z = X + (1's Complement of Y) + 1**. Hence Z = X – Y.



UNSIGNED MULTIPLICATION: CONVENTIONAL METHOD / PENCIL–PAPER METHOD

1) The Conventional (Pencil-Paper) method is used to **multiply two unsigned numbers**.
2) When we multiply two **"N-bit"** numbers, the answer is "**2 x N**" bits.
3) Three registers A, Q and M, are used for this process.
4) **Q** contains the **Multiplier** and **M** contains the **Multiplicand**.
5) **A** (**Accumulator**) is initialized with 0.
6) At the end of the operation, the **Result** will be stored in (**A & Q**) combined.
7) The process involves **addition** and **shifting**.

Algorithm:

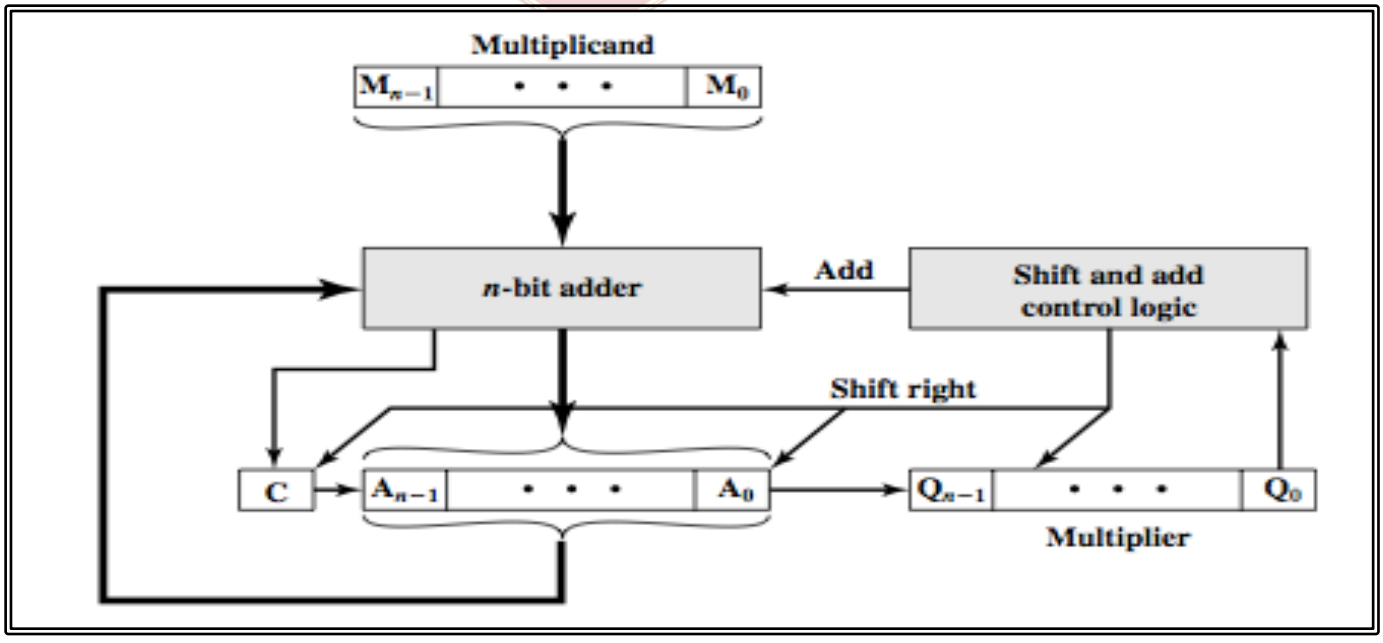
The **number of steps** required is equal to the **number of bits in the multiplier**.

1) At each step, **examine** the current **multiplier bit** starting from the **LSB**.
2) If the current **multiplier bit is "1"**, then the **Partial-Product** is the **Multiplicand** itself.
3) If the current **multiplier bit is "0"**, then the **Partial-Product** is the **Zero**.
4) At each step, **ADD the Partial-Product to the Accumulator**.
5) Now **Right-Shift the Result** produced so far (**A & Q combined**).

Repeat steps 1 to 5 for **all bits** of the multiplier.

The **final answer** will be in **A & Q** combined.

Circuit Diagram for Unsigned Multiplication



Example: 7 x 6 = 42

```
                    0  1  1  1   … Multiplicand (7)
          X         0  1  1  0   … Multiplier (6)
-------------------------------------------------
                    0  0  0  0   … Partial-Product
                 0  1  1  1  X              ”
              0  1  1  1  X  X              ”
+ 0  0  0  0  X  X  X              ”
-------------------------------------------------
  0  1  0  1  0  1  0   … Result (42)
-------------------------------------------------
```

Step	C Carry	A Accumulator	Q Multiplier	M Multiplicand	Explanation
	0	0000	0110	0111	*Initial Value*
1	0 0	0000 0000	0110 0011		*Current Multiplier bit is “0” so ADD “0” to Accumulator and Right-Shift*
2	0 0	0111 0011	0011 1001		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
3	0 0	1010 0101	1001 0100		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
4	0 0	0101 0010 FINAL ANSWER :-)	0100 1010		*Current Multiplier bit is “0” so ADD “0” to Accumulator and Right-Shift*
For more detailed explanation refer Bharat Sir’s Lecture Notes or Call Bharat Sir on 9820408217.					

Example: 15 x 15 = 225

```
                    1   1   1   1   … Multiplicand (15)
          X         1   1   1   1   … Multiplier (15)
-------------------------------------------------------
                    1   1   1   1   … Partial-Product
                1   1   1   1   X               ”
            1   1   1   1   X   X               ”
        1   1   1   1   X   X   X               ”
+   1   1   1   1   X   X   X   X               ”
-------------------------------------------------------
1   1   1   0   0   0   0   1   … Result (225)
-------------------------------------------------------
```

Step	C Carry	A Accumulator	Q Multiplier	M Multiplicand	Explanation
	0	0000	1111	1111	*Initial Value*
1	0 0	1111 0111	1111 1111		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
2	1 0	0110 1011	1111 0111		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
3	1 0	1010 1101	0111 0011		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
4	1 0	1100 1110 FINAL ANSWER :-)	0011 0001		*Current Multiplier bit is “1” so ADD Multiplicand to Accumulator and Right-Shift*
For more detailed explanation refer Bharat Sir’s Lecture Notes or Call Bharat Sir on 9820408217.					

SIGNED MULTIPLICATION: BOOTH'S ALGORITHM

1) Booth's Algorithm is used to **multiply two SIGNED numbers**.
2) When we multiply two **"N-bit"** numbers, the answer is "**2 x N**" bits.
3) Three registers A, Q and M, are used for this process.
4) **Q** contains the **Multiplier** and **M** contains the **Multiplicand**.
5) **A** (**Accumulator**) is initialized with 0.
6) At the end of the operation, the **Result** will be stored in (**A & Q**) combined.
7) The process involves **addition, subtraction** and **shifting**.

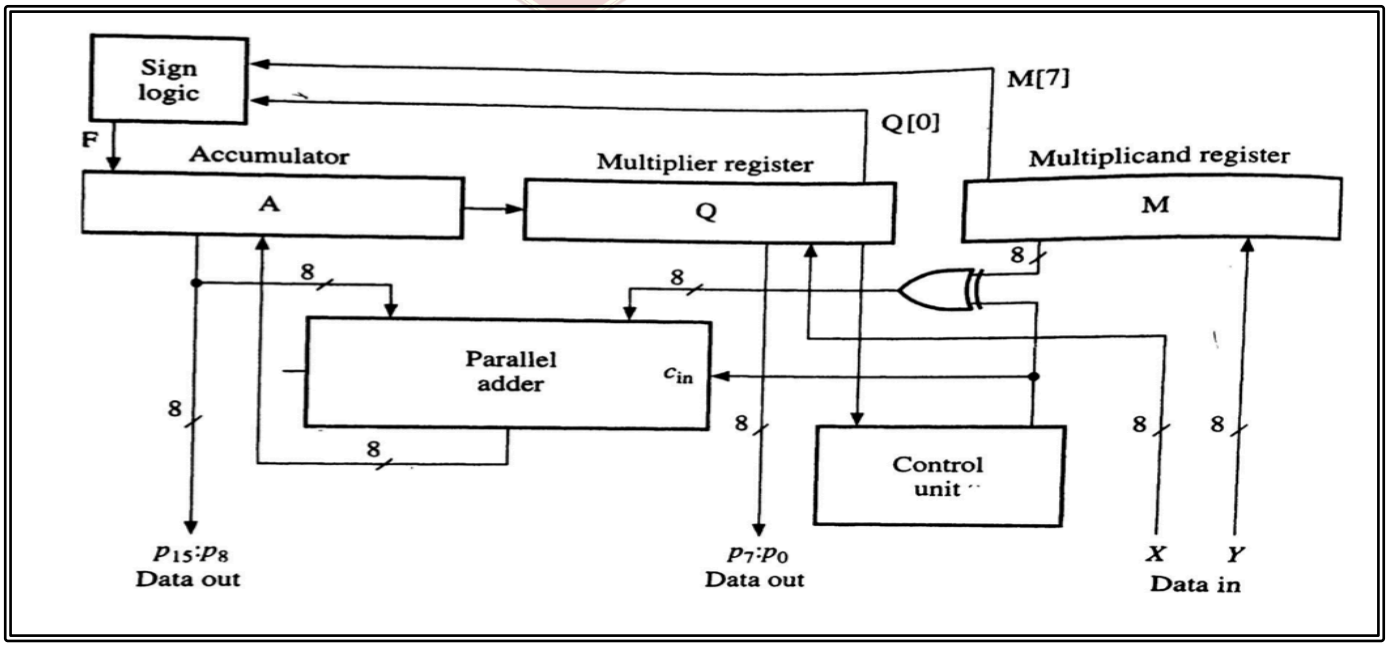
Algorithm:

The **number of steps** required is equal to the **number of bits in the multiplier**.
At the beginning, consider an **imaginary "0" beyond LSB of Multiplier**

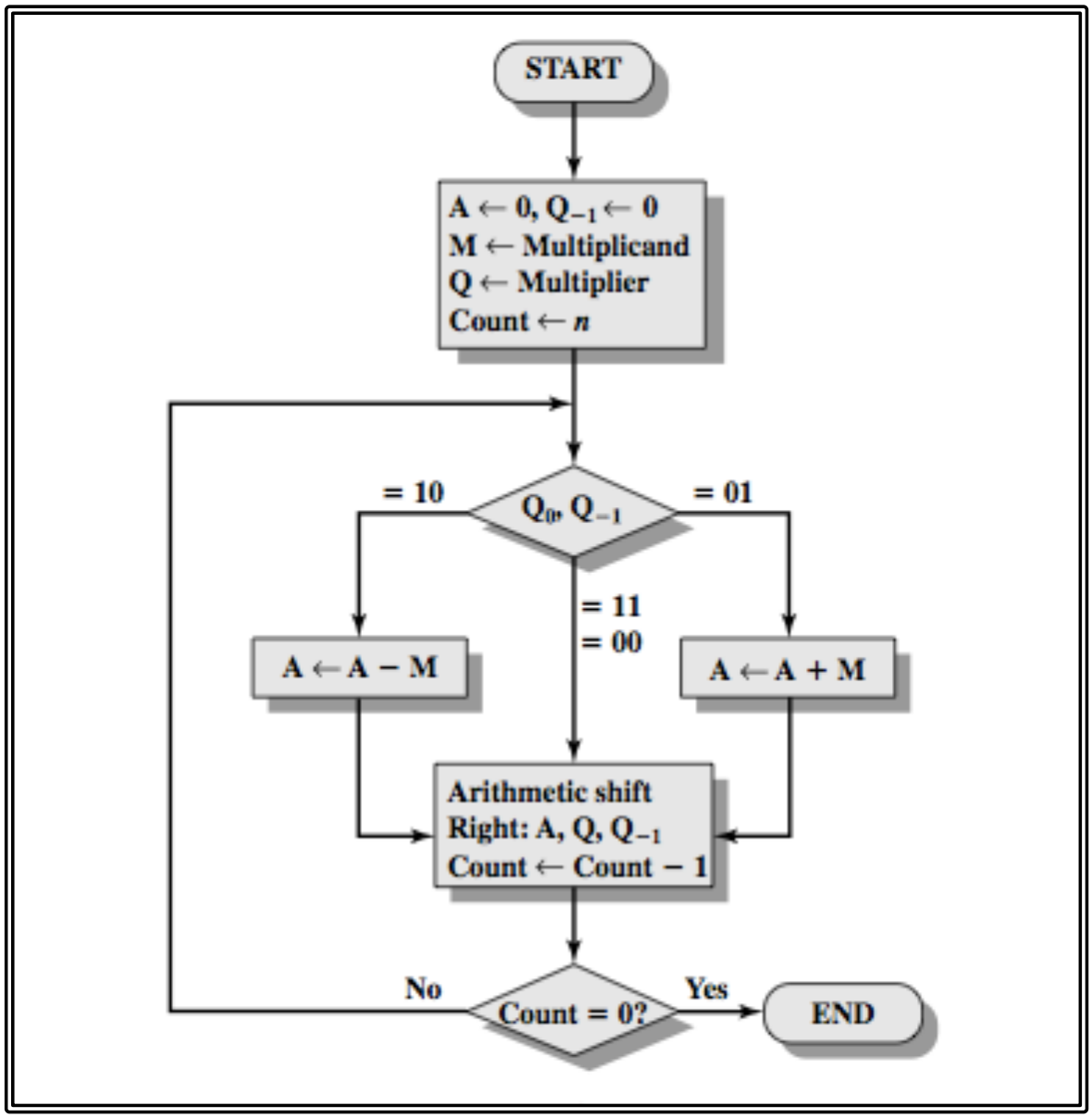
1) At each step, **examine two adjacent Multiplier bits** from **Right to Left**.
2) If the transition is from **"0 to 1"** then **Subtract M** from **A** and **Right-Shift** (A & Q) combined.
3) If the transition is from **"1 to 0"** then **ADD M** to **A** and **Right-Shift**.
4) If the transition is from **"0 to 0"** then **simply Right-Shift**.
5) If the transition is from **"1 to 1"** then **simply Right-Shift**.

Repeat steps 1 to 5 for **all bits** of the multiplier.
The **final answer** will be in **A & Q** combined.

Circuit Diagram for Signed Multiplication using Booth's Algorithm



FLOWCHART FOR BOOTH'S ALGORITHM



Example: 7 x 6 = 42

Multiplicand (M): **7 = 0111**. **-7 = 1001** (Two's Complement Form)
Multiplier (Q): **6 = 0110**. **-6 = 1010** (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	0000	0110	0	**0111**
1) (0 ← 0) No Add or Sub Right-Shift	0000 0000	0110 0011	0 0	
2) (1 ← 0) Perform **(A - M)** Right-Shift	1001 1100	0011 1001	0 1	
3) (1 ← 1) No Add or Sub Right-Shift	1100 1110	1001 0100	1 1	
4) (0 ← 1) Perform **(A + M)** Right-Shift	0101 **0010** **FINAL ANSWER :-)**	0100 **1010**	1 **0**	

For more detailed explanation refer Bharat Sir's Lecture Notes or Call Bharat Sir on 9820408217.

Example: 5 x 7 = 35

Multiplicand (M): **5 = 0101**. **-5 = 1011** (Two's Complement Form)
Multiplier (Q): **7 = 0111**. **-7 = 1001** (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	0000	0111	0	**0101**
1) (1 ← 0) Perform **(A - M)** Right-Shift	1011 1101	0111 1011	0 1	
2) (1 ← 1) No Add or Sub Right-Shift	1101 1110	1011 1101	1 1	
3) (1 ← 1) No Add or Sub Right-Shift	1110 1111	1101 0110	1 1	
4) (0 ← 1) Perform **(A + M)** Right-Shift	0100 0010	0110 0011	1 **0**	
	FINAL ANSWER :-)			
For more detailed explanation refer Bharat Sir's Lecture Notes or Call Bharat Sir on 9820408217.				

Example: 9 x 10 = 90

Multiplicand (M): **9 = 01001**. **-9 = 10111** (Two's Complement Form)
Multiplier (Q): **10 = 01010**. **-10 = 10110** (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	00000	01010	**0**	**01001**
1) (0 ← 0) No Add or Sub Right-Shift	00000 00000	01010 00101	**0** **0**	
2) (1 ← 0) Perform **(A - M)** Right-Shift	10111 11011	00101 10010	**0** **1**	
3) (0 ← 1) Perform **(A + M)** Right-Shift	00100 00010	10010 01001	**1** **0**	
4) (1 ← 0) Perform **(A - M)** Right-Shift	11001 11100	01001 10100	**0** **1**	
5) (0 ← 1) Perform **(A + M)** Right-Shift	00101 00010 **FINAL ANSWER :-)**	10100 11010	**1** **0**	
Practice more sums by yourself. In case of doubts call #bharatsir at 9820408217.				

Expected Ans: 90 = 00010 11010

Example: -9 x 10 = -90

Multiplicand (M): **-9 = 10111** **9 = 01001**. (Two's Complement Form)
Multiplier (Q): **10 = 01010**. **-10 = 10110** (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	00000	01010	**0**	**10111**
1) (0 ← 0) No Add or Sub Right-Shift	00000 00000	01010 00101	**0** **0**	
2) (1 ← 0) Perform **(A - M)** Right-Shift	01001 00100	00101 10010	**0** **1**	
3) (0 ← 1) Perform **(A + M)** Right-Shift	11011 11101	10010 11001	**1** **0**	
4) (1 ← 0) Perform **(A - M)** Right-Shift	00110 00011	11001 01100	**0** **1**	
5) (0 ← 1) Perform **(A + M)** Right-Shift	11010 **11101**	01100 **00110**	**1** **0**	
	FINAL ANSWER :-)			
Practice more sums by yourself. In case of doubts call #bharatsir at 9820408217.				

Expected Ans: -90 = 11101 00110

Example: 9 x -10 = -90

Multiplicand (M): **9 = 01001** **-9 = 10111**. (Two's Complement Form)
Multiplier (Q): **-10 = 10110** **10 = 01010**. (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	00000	10110	0	01001
1) (0 ← 0) No Add or Sub Right-Shift	00000 00000	10110 01011	0 0	
2) (1 ← 0) Perform **(A - M)** Right-Shift	10111 11011	01011 10101	0 1	
3) (1 ← 1) No Add or Sub Right-Shift	11011 11101	10101 11010	1 1	
4) (0 ← 1) Perform **(A + M)** Right-Shift	00110 00011	11010 01101	1 0	
5) (1 ← 0) Perform **(A - M)** Right-Shift	11010 **11101** **FINAL ANSWER :-)**	01101 **00110**	0 **1**	
Practice more sums by yourself. In case of doubts call #bharatsir at 9820408217.				

Expected Ans: -90 = 11101 00110

Example: -9 x -10 = 90

Multiplicand (M): **-9 = 10111** **9 = 01001**. (Two's Complement Form)
Multiplier (Q): **-10 = 10110** **10 = 01010**. (Two's Complement Form)

Step	A Accumulator	Q Multiplier	Q_{-1} Imaginary	M Multiplicand
Initial	00000	10110	0	10111
1) (0 ← 0) No Add or Sub Right-Shift	00000 00000	10110 01011	0 0	
2) (1 ← 0) Perform **(A - M)** Right-Shift	01001 00100	01011 10101	0 1	
3) (1 ← 1) No Add or Sub Right-Shift	00100 00010	10101 01010	1 1	
4) (0 ← 1) Perform **(A + M)** Right-Shift	11001 11100	01010 10101	1 0	
5) (1 ← 0) Perform **(A - M)** Right-Shift	00101 **00010**	10101 **11010**	0 **0**	
	FINAL ANSWER :-)			
Practice more sums by yourself. In case of doubts call #bharatsir at 9820408217.				

Expected Ans: 90 = 00010 11010

RESTORING DIVISION

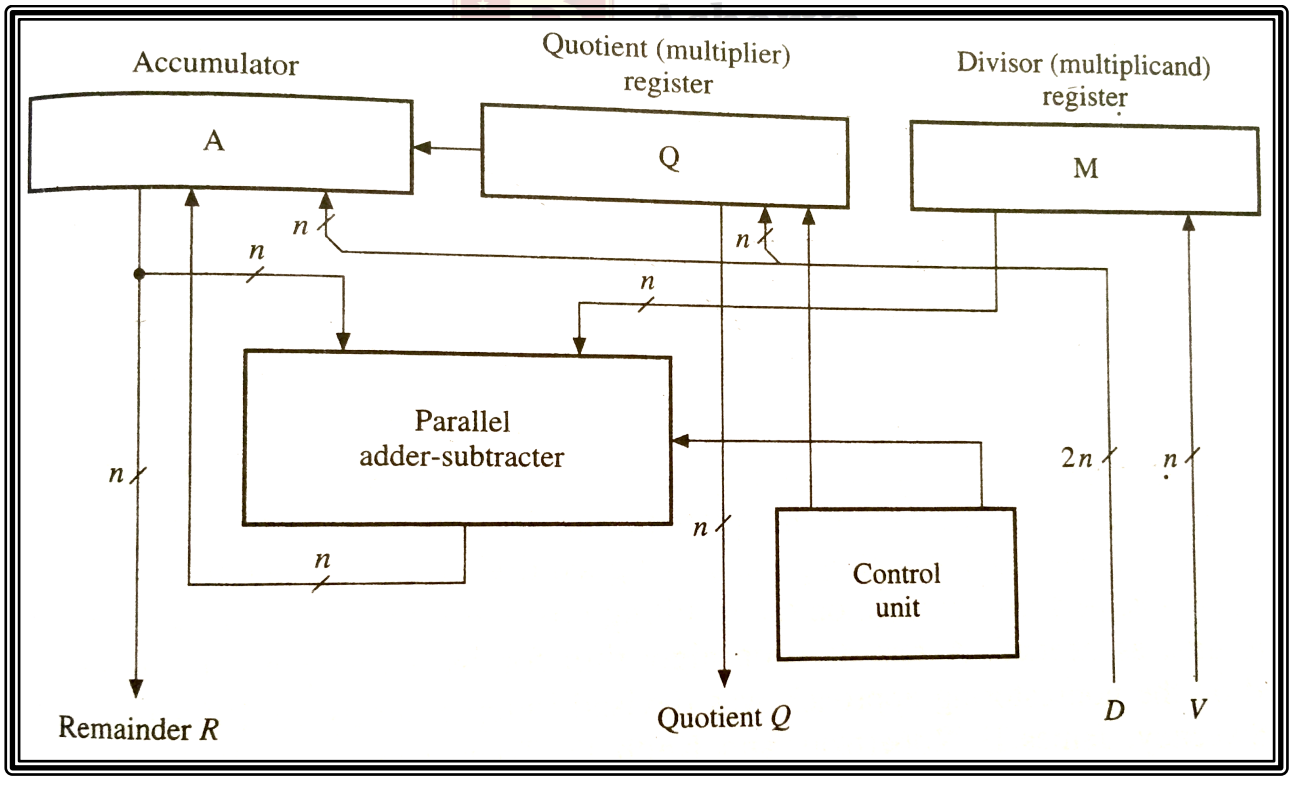
1) Let Q register hold the divided, M register holds the divisor and A register is 0.
2) On completion of the algorithm, Q will get the quotient and A will get the remainder.

Algorithm:

The **number of steps** required is equal to the **number of bits in the Dividend**.

1) At each step, **left shift the dividend by 1 position**.
2) **Subtract the divisor from A** (perform A - M).
3) If the **result is positive** then the step is said to be **"Successful"**.
   In this case **quotient bit will be "1"** and **Restoration is NOT Required**.
4) If the **result is negative** then the step is said to be **"Unsuccessful"**.
   In this case **quotient bit will be "0".**
   Here Restoration is performed by adding back the divisor.
   Hence the method is called Restoring Division.

Repeat steps 1 to 4 for **all bits** of the Dividend.



Example: (6) / (4)

Dividend (Q) = 6
Divisor (M) = 4
Accumulator (A) = 0

6 = 0110 4 = 0100
-6 = 1010 -4 = 1100

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (6)	DIVISOR M (4)
Initial Values	0000	0110	0100
Step 1:			
Left-Shift	0000	110_	
A – M	+1100		
Unsuccessful (-ve)	1100		
Restoration	0000	1100	
Step 2:			
Left-Shift	0001	100_	
A – M	+1100		
Unsuccessful (-ve)	1101		
Restoration	0001	1000	
Step 3:			
Left-Shift	0011	000_	
A – M	+1100		
Unsuccessful (-ve)	1111		
Restoration	0011	0000	
Step 4:			
Left-Shift	0110	000_	
A – M	+1100		
Successful (+ve)	0010		
No Restoration	0000	0001	
	Remainder(0)	**Quotient(1)**	

Example: (19) / (7)

Dividend (Q) = 19 Divisor (M) = 7 Accumulator (A) = 0

19 = 010011 7 = 000111
-19 = 101101 -7 = 111001

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (19)	DIVISOR M (7)
Initial Values	000000	010011	000111
Step 1:			
Left-Shift	000000	10011_	
A – M	+111001		
Unsuccessful (-ve)	111001		
Restoration	000000	100110	
Step 2:			
Left-Shift	000001	00110_	
A – M	+111001		
Unsuccessful (-ve)	111010		
Restoration	000001	001100	
Step 3:			
Left-Shift	000010	01100_	
A – M	+111001		
Unsuccessful (-ve)	111011		
Restoration	000010	011000	
Step 4:			
Left-Shift	000100	11000_	
A – M	+111001		
Unsuccessful (-ve)	111101		
Restoration	000100	110000	
Step 5:			
Left-Shift	001001	10000_	
A – M	+111001		
Successful (+ve)	000010		
No Restoration	000010	100001	
Step 6:			
Left-Shift	000101	00001_	
A – M	+111001		
Unsuccessful (-ve)	111110		
Restoration	000101	000010	
	Remainder(5)	Quotient(2)	

Example: (23) / (3)

Dividend (Q) = 23 Divisor (M) = 3 Accumulator (A) = 0

23 = 010111 3 = 000011
-23 = 101001 -3 = 111101

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (23)	DIVISOR M (3)
Initial Values	000000	010111	000011
Step 1:			
Step 2:			
Step 3:			
Step 4:			
Step 5:			
Step 6:			
	Remainder(2)	Quotient(7)	

NON-RESTORING DIVISION

1) Let Q register hold the divided, M register holds the divisor and A register is 0.
2) On completion of the algorithm, Q will get the quotient and A will get the remainder.

Algorithm:

The **number of steps** required is equal to the **number of bits in the Dividend**.

1) At each step, **left shift the dividend by 1 position**.
2) **Subtract the divisor from A** (perform A - M).
3) If the **result is positive** then the step is said to be **"Successful"**.
   In this case **quotient bit will be "1"** and **Restoration is NOT Required**.
   The **Next Step** will also be **Subtraction**.
4) If the **result is negative** then the step is said to be **"Unsuccessful"**.
   In this case **quotient bit will be "0".**
   Here Restoration is NOT Performed.
   Instead the next step will be ADDITION in place of subtraction.
   As restoration is not performed, the method is called Non-Restoring Division.

Repeat steps 1 to 4 for **all bits** of the Dividend.

Note: Restoration may only be performed in the FINAL step, if it is unsuccessful, as there is no next step thereafter. In such a case, restoration is performed by adding back the divisor.

Example: (7) / (5)

Dividend (Q) = 7
Divisor (M) = 5
Accumulator (A) = 0

7 = 0111 5 = 0101
-7 = 1001 -5 = 1011

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (7)	DIVISOR M (5)
Initial Values	0000	0111	0101
Step 1:			
Left-Shift	0000	111_	
A – M	+1011		
Unsuccessful (-ve)	1011	1110	
Next Step: "Add"			
Step 2:			
Left-Shift	0111	110_	
A + M	+0101		
Unsuccessful (-ve)	1100	1100	
Next Step: "Add"			
Step 3:			
Left-Shift	1001	100_	
A + M	+0101		
Unsuccessful (-ve)	1110	1000	
Next Step: "Add"			
Step 4:			
Left-Shift	1101	000_	
A + M	+0101		
Successful (+ve)	0010	0001	
	Remainder(2)	Quotient(1)	

Example: (7) / (3)

Dividend (Q) = 7
Divisor (M) = 3
Accumulator (A) = 0

7 = 0111 3 = 0011
-7 = 1001 -3 = 1101

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (7)	DIVISOR M (3)
Initial Values	0000	0111	0011
Step 1:			
Left-Shift	0000	111_	
A – M	+1101		
Unsuccessful (-ve)	1101	1110	
Next Step: "Add"			
Step 2:			
Left-Shift	1011	110_	
A + M	+0011		
Unsuccessful (-ve)	1110	1100	
Next Step: "Add"			
Step 3:			
Left-Shift	1101	100_	
A + M	+0011		
Successful (+ve)	0000	1001	
Next Step: "Sub"			
Step 4:			
Left-Shift	0001	001_	
A - M	+1101		
Unsuccessful (-ve)	1110		
Restore: (Add back divisor)	0001	0010	
	Remainder(1)	**Quotient(2)**	

Example: (19) / (4)

Dividend (Q) = 19 Divisor (M) = 4 Accumulator (A) = 0

19 = 010011 4 = 000100
-19 = 101101 -4 = 111100

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR	DIVIDEND	DIVISOR
	A (0)	Q (19)	M (4)
Initial Values	000000	010011	000100
Step 1:			
Left-Shift	000000	10011_	
A – M	+111100		
Unsuccessful (-ve)	111100	100110	
Next Step: "Add"			
Step 2:			
Left-Shift	111001	00110_	
A + M	+000100		
Unsuccessful (-ve)	111101	001100	
Next Step: "Add"			
Step 3:			
Left-Shift	111010	01100_	
A + M	+000100		
Unsuccessful (-ve)	111110	011000	
Next Step: "Add"			
Step 4:			
Left-Shift	111100	11000_	
A + M	+000100		
Successful (+ve)	000000	110001	
Next Step: "Sub"			
Step 5:			
Left-Shift	000001	10001_	
A – M	+111100		
Unsuccessful (-ve)	111101	100010	
Next Step: "Add"			
Step 6:			
Left-Shift	111011	00010_	
A + M	+000100		
Unsuccessful (-ve)	111111		
Restore: (Add back divisor)	000011	000100	
	Remainder(3)	Quotient(4)	

Example: (23) / (3)

Dividend (Q) = 23 Divisor (M) = 3 Accumulator (A) = 0

23 = 010111 3 = 000011
-23 = 101001 -3 = 111101

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (0)	DIVIDEND Q (23)	DIVISOR M (3)
Initial Values	000000	010111	000011
Step 1:			
Step 2:			
Step 3:			
Step 4:			
Step 5:			
Step 6:			
	Remainder(2)	Quotient(7)	

RESTORING DIVISION FOR SIGNED NUMBERS

1) Let M register hold the divisor, Q register hold the divided.
2) A register should be the signed extension of Q.
3) On completion of the algorithm, Q will get the quotient and A will get the remainder.

Algorithm:

The **number of steps** required is equal to the **number of bits in the Dividend**.

1) At each step, **left shift the dividend by 1 position**.
2) If Sign of A and M is the same then **Subtract the divisor from A** (perform A - M),
   Else **Add M to A**
3) After the operation,
   If **Sign of A remains the same** or the **dividend** (in A and Q) **becomes zero**,
   then the step is said to be **"Successful"**.
   In this case **quotient bit will be "1"** and **Restoration is NOT Required**.
4) If **Sign of A changes**, then the step is said to be **"Unsuccessful"**.
   In this case **quotient bit will be "0".**
   Here Restoration is Performed.
   Hence, the method is called Restoring Division.

Repeat steps 1 to 4 for **all bits** of the Dividend.

Note: The result of this algorithm is such that, the quotient will always be positive and the remainder will get the same sign as the dividend.

Example: (5) / (3)

5 = 0101 3 = 0011
-5 = 1011 -3 = 1101

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (Sign Extension)	DIVIDEND Q (5)	DIVISOR M (3)
Initial Values	0000	0101	0011
Step 1:			
Left-Shift	0000	101_	
Sign (A, M) Same: A - M	+1101		
Sign Changes: Unsuccessful	1101		
Restore	0000	1010	
Step 2:			
Left-Shift	0001	010_	
Sign (A, M) Same: A - M	+1101		
Sign Changes: Unsuccessful	1110		
Restore	0001	0100	
Step 3:			
Left-Shift	0010	100_	
Sign (A, M) Same: A – M	+1101		
Sign Changes: Unsuccessful	1111		
Restore	0010	1000	
Step 4:			
Left-Shift	0101	000_	
Sign (A, M) Same: A - M	+1101		
Sign still Same: Successful	0010		
Restore not required	0010	0001	
	Remainder(2)	Quotient(1)	

Example: (-19) / (7)

19 = 010011 7 = 000111
-19 = 101101 -7 = 111001

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR	DIVIDEND	DIVISOR
	A (Sign Extension)	Q (-19)	M (7)
Initial Values	111111	101101	000111
Step 1:			
Left-Shift	111111	01101_	
Sign (A, M) Different: A + M	+000111		
Sign Changes: Unsuccessful	000110		
Restore	111111	011010	
Step 2:			
Left-Shift	111110	11010_	
Sign (A, M) Different: A + M	+000111		
Sign Changes: Unsuccessful	000101		
Restore	111110	110100	
Step 3:			
Left-Shift	111101	10100_	
Sign (A, M) Different: A + M	+000111		
Sign Changes: Unsuccessful	000100		
Restore	111101	101000	
Step 4:			
Left-Shift	111011	01000_	
Sign (A, M) Different: A + M	+000111		
Sign Changes: Unsuccessful	000010		
Restore	111011	010000	
Step 5:			
Left-Shift	110110	10000_	
Sign (A, M) Different: A + M	+000111		
Sign still Same: Successful	111101		
Restore not required	111101	100001	
Step 6:			
Left-Shift	111011	00001_	
Sign (A, M) Different: A + M	+000111		
Sign Changes: Unsuccessful	000010		
Restore	111011	000010	
	Remainder(-5)	Quotient(2)	

Example: (-8) / (-4)

8 = 01000 4 = 00100
-8 = 11000 -4 = 11100

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR	DIVIDEND	DIVISOR
	A (Sign Extension)	Q (-8)	M (-4)
Initial Values	11111	11000	11100
Step 1:			
Left-Shift	11111	1000_	
Sign (A, M) Same: A - M	+00100		
Sign Changes: Unsuccessful	00011		
Restore	11111	10000	
Step 2:			
Left-Shift	11111	0000_	
Sign (A, M) Same: A - M	+00100		
Sign Changes: Unsuccessful	00011		
Restore	11111	00000	
Step 3:			
Left-Shift	11110	0000_	
Sign (A, M) Same: A - M	+00100		
Sign Changes: Unsuccessful	00010		
Restore	11110	00000	
Step 4:			
Left-Shift	11100	0000_	
Sign (A, M) Same: A - M	+00100		
Dividend(A,Q)=0: Successful	00000		
Restore not required	00000	00001	
Step 5:			
Left-Shift	00000	0001_	
Sign (A, M) Different: A + M	+11100		
Sign Changes: Unsuccessful	11100		
Restore	00000	00010	
	Remainder(0)	**Quotient(2)**	

Example: (-14) / (2)

14 = 01110 2 = 00010
-14 = 10010 -2 = 11110

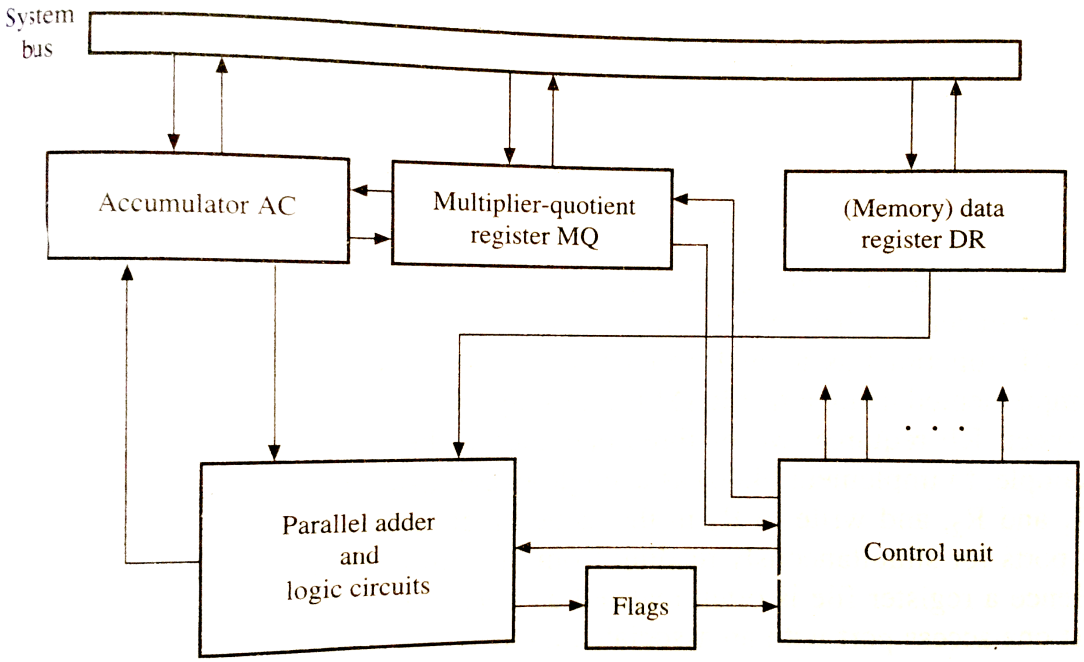
In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR	DIVIDEND	DIVISOR
	A (Sign Extension)	Q (-14)	M (2)
Initial Values	11111	10010	00010
Step 1:			
Left-Shift	11111	0010_	
Sign (A, M) Different: A + M	+00010		
Sign Changes: Unsuccessful	00001		
Restore	11111	00100	
Step 2:			
Left-Shift	11110	0100_	
Sign (A, M) Different: A + M	+00010		
Sign Changes: Unsuccessful	00000	*Note that dividend part in (A, Q) is not zero*	
Restore	11110	01000	
Step 3:	*In case of doubts, call #BharatSir @ 98204 08217*		
Left-Shift	11100	1000_	
Sign (A, M) Different: A + M	+00010		
Sign still Same: Successful	11110		
Restore not required	11110	10001	
Step 4:			
Left-Shift	11101	0001_	
Sign (A, M) Different: A + M	+00010		
Sign still Same: Successful	11111		
Restore not required	11111	00011	
Step 5:			
Left-Shift	11110	0011_	
Sign (A, M) Different: A + M	+00010		
Dividend(A,Q)=0: Successful	00000	*Note that dividend part in (A, Q) is Zero!*	
Restore not required	00000	00111	
	Remainder(0)	**Quotient(7)**	

Example: (14) / (-2)

14 = 01110 2 = 00010
-14 = 10010 -2 = 11110

In case of doubts, Call #BharatSir @ 98204 08217	ACCUMULATOR A (Sign Extension)	DIVIDEND Q (-14)	DIVISOR M (-2)
Initial Values	00000	01110	11110
Step 1:			
Step 2:			
Step 3:	*In case of doubts, call #BharatSir @ 98204 08217*		
Step 4:			
Step 5:			
	Remainder(0)	Quotient(7)	

TYPICAL ALU DESIGN



Addition	AC := AC + DR
Subtraction	AC := AC – DR
Multiplication	AC.MQ := DR × MQ
Division	AC.MQ := MQ/DR
AND	AC := AC ***and*** DR
OR	AC := AC ***or*** DR
EXCLUSIVE-OR	AC := AC ***xor*** DR
NOT	AC := ***not***(AC)

FLOATING POINT NUMBERS

- In some numbers, which have a fractional part, the position of the decimal point is not fixed as the number of bits before (or after) the decimal point may vary.
- **Eg: 0010.01001, 0.0001101, -1001001.01** etc.
- As shown above, the position of the decimal point is not fixed, instead it **"floats"** in the number.

For doubts contact Bharat Sir on 98204 08217

- Such numbers are called Floating Point Numbers.
- Floating Point Numbers are stored in a "Normalized" form.

NORMALIZATION OF A FLOATING POINT NUMBER

- Normalization is the process of shifting the point, left or right, so that there is only one non-zero digit to the left of the point. #Please refer Bharat Sir's Lecture Notes for this ...

Floating Point Number		Normalized Number
01010.01	→	$(-1)^{0} \times 1.\mathbf{01001} \times 2^{3}$
11111.01	→	$(-1)^{0} \times 1.\mathbf{111101} \times 2^{4}$
0.00101	→	$(-1)^{0} \times 1.\mathbf{01} \times 2^{-3}$
-10.01	→	$(-1)^{1} \times 1.\mathbf{001} \times 2^{1}$

Bharat Acharya Education

- As seen above a Normalized Form of a number is:

$$(-1)^{S} \times 1.M \times 2^{E}$$

Where: S = Sign, M = Mantissa and E = Exponent.

- As Normalized numbers are of the 1.M format, the "**1**" is not actually stored, it is instead **assumed**. This saves the storage space by 1 bit for each number.

- Also the Exponent is stored in the biased form by adding an appropriate bias value to it so that -ve exponents can be easily represented.

Advantages of Normalization.

1. Storing all numbers in a standard for makes **calculations easier** and **faster**.
2. By **not storing** the **1** (of 1.M format) for a number, considerable **storage space** is **saved**.
3. The **exponent** is **biased** so there is **no need** for **storing** its **sign bit** (as the biased exponent cannot be -ve).

SHORT REAL FORMAT / SINGLE PRECISION FORMAT / IEEE 754: 32 BIT FORMAT

S	Biased Exponent	Mantissa
(1)	**(8)** **Bias value = 127**	**(23 bits)**

- **32 bits** are used to store the **number**.
- **23 bits** are used for the **Mantissa**.
- **8 bits** are used for the Biased **Exponent**.
- **1 bit** used for the **Sign** of the number.
- The **Bias** value is $(\mathbf{127})_{10}$.
- The range is $\pm\mathbf{1 \times 10^{-38}}$ **to** $\pm\mathbf{3 \times 10^{38}}$ approximately.
- It is called as the **Single Precision Format** for Floating-Point Numbers.

LONG REAL FORMAT / DOUBLE PRECISION FORMAT / IEEE 754: 64 BIT FORMAT

S	Biased Exponent	Mantissa
(1)	**(11)** **Bias value = 1023**	**(52 bits)**

Bharat Acharya Education

- **64 bits** are used to store the **number**.
- **52 bits** are used for the **Mantissa**.
- **11 bits** are used for the Biased **Exponent**.
- **1 bit** used for the **Sign** of the number.
- The **Bias** value is $(\mathbf{1023})_{10}$.
- The range is $\pm\mathbf{10^{-308}}$ **to** $\pm\mathbf{10^{308}}$ approximately.
- It is called as the **Double Precision Format** for Floating-Point Numbers.

Numericals on Floating Point Number Representation

1) Convert 2A3BH into Short Real and Temp Real formats {Exam question}

Short real:

Converting the number into binary we get:
0010 1010 0011 1011

Normalizing the number we get:
$(-1)^{0}$ **x** 1.**0101000111011** x 2^{13}
Here S = 0; M = 0101000111011; True Exponent = 13.

Bias value for Short Real format is 127:
Biased Exponent (BE) = True Exponent + Bias
= 13 + 127
= 140.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (1000 1100)

Representing in the required format we get:

0	**10001100**	**01010001101100…**
S	Biased Exp	Mantissa
(1)	(8)	(23)

Converting the number into hexadecimal form we get:
4628EC00H … 32 bits.

Temp real:

Bias value for Temp Real format is 16383:
Biased Exponent (BE) = True Exponent + Bias
= 13 + 16383
= 16396.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (100 0000 0000 1100)

Representing in the required format we get:

0	100000000001100	**1**01010001101100…
S	Biased Exp	Mantissa
(1)	(15)	(64)

Converting the number into hexadecimal form we get:
400C A8EC 0000 0000 0000H … 80 bits.

2) Convert $(12.125)_d$ into Temp Real format {Exam question}

Temp real:

Converting the number into binary we get:
1100.001 For doubts contact Bharat Sir on 98204 08217

Normalizing the number we get:
$(-1)^{\mathbf{0}}$ **x** 1.**100001** x $2^{\mathbf{3}}$
Here S = 0; M = 100001; True Exponent = 3.

Bias value for Temp Real format is 16383:
Biased Exponent (BE) = True Exponent + Bias
= 3 + 16383
= 16386.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (100 0000 0000 0010)

Representing in the required format we get:

0	100000000000010	**1**10000100000…
S	Biased Exp	Mantissa
(1)	(8)	(23)

Converting the number into hexadecimal form we get:
4002 C200 0000 0000 0000H … 80 bits.

FLOATING POINT NUMBERS

- In some numbers, which have a fractional part, the position of the decimal point is not fixed as the number of bits before (or after) the decimal point may vary.
- **Eg: 0010.01001, 0.0001101, -1001001.01** etc.
- As shown above, the position of the decimal point is not fixed, instead it **"floats"** in the number.

For doubts contact Bharat Sir on 98204 08217

- Such numbers are called Floating Point Numbers.
- Floating Point Numbers are stored in a "Normalized" form.

NORMALIZATION OF A FLOATING POINT NUMBER

- Normalization is the process of shifting the point, left or right, so that there is only one non-zero digit to the left of the point. #Please refer Bharat Sir's Lecture Notes for this ...

Floating Point Number		Normalized Number
01010.01	→	$(-1)^{0} \times 1.\mathbf{01001} \times 2^{3}$
11111.01	→	$(-1)^{0} \times 1.\mathbf{111101} \times 2^{4}$
0.00101	→	$(-1)^{0} \times 1.\mathbf{01} \times 2^{-3}$
-10.01	→	$(-1)^{1} \times 1.\mathbf{001} \times 2^{1}$

Bharat Acharya Education

- As seen above a Normalized Form of a number is:

$$(-1)^{S} \times 1.M \times 2^{E}$$

Where: S = Sign, M = Mantissa and E = Exponent.

- As Normalized numbers are of the 1.M format, the "**1**" is not actually stored, it is instead **assumed**. This saves the storage space by 1 bit for each number.

- Also the Exponent is stored in the biased form by adding an appropriate bias value to it so that -ve exponents can be easily represented.

Advantages of Normalization.

1. Storing all numbers in a standard for makes **calculations easier** and **faster**.
2. By **not storing** the **1** (of 1.M format) for a number, considerable **storage space** is **saved**.
3. The **exponent** is **biased** so there is **no need** for **storing** its **sign bit** (as the biased exponent cannot be -ve).

SHORT REAL FORMAT / SINGLE PRECISION FORMAT / IEEE 754: 32 BIT FORMAT

S	Biased Exponent	Mantissa
(1)	**(8)** **Bias value = 127**	**(23 bits)**

- **32 bits** are used to store the **number**.
- **23 bits** are used for the **Mantissa**.
- **8 bits** are used for the Biased **Exponent**.
- **1 bit** used for the **Sign** of the number.
- The **Bias** value is $(\mathbf{127})_{10}$.
- The range is $\pm\mathbf{1 \times 10^{-38}}$ **to** $\pm\mathbf{3 \times 10^{38}}$ approximately.
- It is called as the **Single Precision Format** for Floating-Point Numbers.

LONG REAL FORMAT / DOUBLE PRECISION FORMAT / IEEE 754: 64 BIT FORMAT

S	Biased Exponent	Mantissa
(1)	**(11)** **Bias value = 1023**	**(52 bits)**

Bharat Acharya Education

- **64 bits** are used to store the **number**.
- **52 bits** are used for the **Mantissa**.
- **11 bits** are used for the Biased **Exponent**.
- **1 bit** used for the **Sign** of the number.
- The **Bias** value is $(\mathbf{1023})_{10}$.
- The range is $\pm\mathbf{10^{-308}}$ **to** $\pm\mathbf{10^{308}}$ approximately.
- It is called as the **Double Precision Format** for Floating-Point Numbers.

Numericals on Floating Point Number Representation

1) Convert 2A3BH into Short Real and Temp Real formats {Exam question}

Short real:

Converting the number into binary we get:
0010 1010 0011 1011

Normalizing the number we get:
$(-1)^{0}$ **x** 1.**0101000111011** x 2^{13}
Here S = 0; M = 0101000111011; True Exponent = 13.

Bias value for Short Real format is 127:
Biased Exponent (BE) = True Exponent + Bias
= 13 + 127
= 140.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (1000 1100)

Representing in the required format we get:

0	**10001100**	**01010001101100…**
S	Biased Exp	Mantissa
(1)	(8)	(23)

Converting the number into hexadecimal form we get:
4628EC00H … 32 bits.

Temp real:

Bias value for Temp Real format is 16383:
Biased Exponent (BE) = True Exponent + Bias
= 13 + 16383
= 16396.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (100 0000 0000 1100)

Representing in the required format we get:

0	100000000001100	**1**01010001101100…
S	Biased Exp	Mantissa
(1)	(15)	(64)

Converting the number into hexadecimal form we get:
400C A8EC 0000 0000 0000H … 80 bits.

2) Convert $(12.125)_d$ into Temp Real format {Exam question}

Temp real:

Converting the number into binary we get:
1100.001 For doubts contact Bharat Sir on 98204 08217

Normalizing the number we get:
$(-1)^{\mathbf{0}}$ **x** 1.**100001** x $2^{\mathbf{3}}$
Here S = 0; M = 100001; True Exponent = 3.

Bias value for Temp Real format is 16383:
Biased Exponent (BE) = True Exponent + Bias
= 3 + 16383
= 16386.

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (100 0000 0000 0010)

Representing in the required format we get:

0	100000000000010	**1**10000100000…
S	Biased Exp	Mantissa
(1)	(8)	(23)

Converting the number into hexadecimal form we get:
4002 C200 0000 0000 0000H … 80 bits.

FLOATING POINT ALU

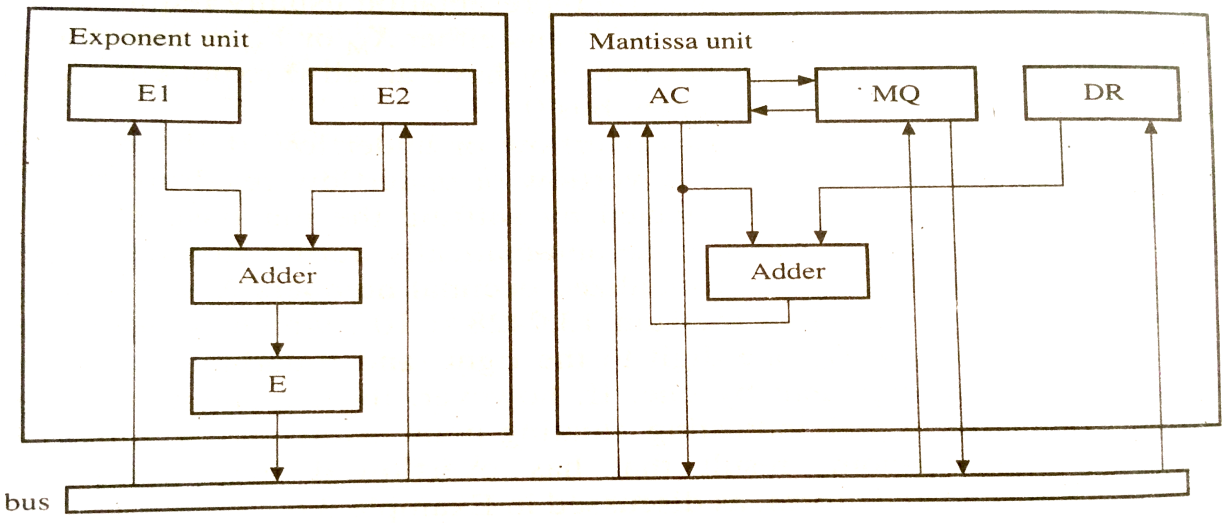
While performing arithmetic operations, a Floating point number is treated as two fixed point numbers: Exponent and Mantissa.

Consider X and Y as two Floating point numbers.
$X = X_M.2^{X_E}$
$Y = Y_M.2^{Y_E}$

Then the arithmetic operations on these two numbers will be performed as:

OPERATION	MANTISSA	EXPONENT
Addition	Add	Equalize
Subtraction	Sub	Equalize
Multiplication	Mul	Add
Division	Div	Sub

From the above table it is evident that Mantissa and Exponent are dealt with, in different ways. Hence a Floating Point ALU has two units internally: Exponent Unit and Mantissa Unit.



FLOATING POINT ADDITION ALGORITHM

```
LOAD:       E1 ← XE,  E2 ← YE;  ...          {Exponents}
            AC ← XM,  DR ← YM; ...           {Mantissas}
            Error ← 0,  AC_Overflow ← 0; ...  {Error Variables}

{Compare and Equalize}
COMPARE:    E ← E1 – E2;

EQUALIZE:   If (E < 0) then
                  AC ← right-shift (AC);
                  E ← E + 1;
                  go to Equalize;
            Else
            If (E > 0) then
                  DR ← right-shift (DR);
                  E ← E – 1;
                  go to Equalize;

{Add the mantissas}
ADD:        AC ← AC + DR;
            E ← Max (E1, E2);

{Adjust for overflow}
OVERFLOW:   If (AC_Overflow = 1) then
                  If (E = Emax) then go to ERROR;
                  AC ← right-shift (AC);
                  E ← E + 1;
                  go to END;

{Adjust for Zero result}
ZERO:       If (AC = 0) then
                  E ← 0;

{Normalize the result}
NORMALIZE:  If AC is normalized then
                  go to END;

UNDERFLOW:        If E > EMIN then
                  AC ← left-shift (AC);
                  E ← E – 1;
                  go to Normalize;

{Set error flag}
ERROR:            Error ← 1;

{End the program}
END:              End of process.
```

PIPELINING

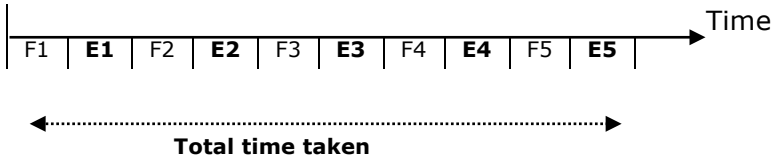
Overlapping different stages of an instruction is called pipelining.

An instruction requires several steps which mainly involve fetching, decoding and execution.
If these steps are performed one after the other, they will take a long time.
As processors became faster, several of these steps started to get overlapped, resulting in faster processing. This is done by a mechanism called pipelining.

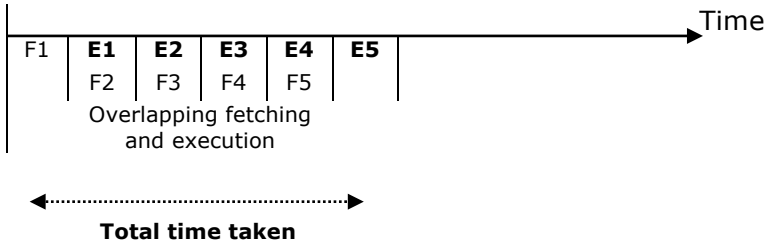
2 STAGE PIPELINING - 8086

Here the instruction process in divided into two stages of fetching and execution.
Fetching of the next instruction takes place while the current instruction is being executed. Hence two instructions are being processed at any point of time.

NON-PIPELINED PROCESSOR EG: 8085

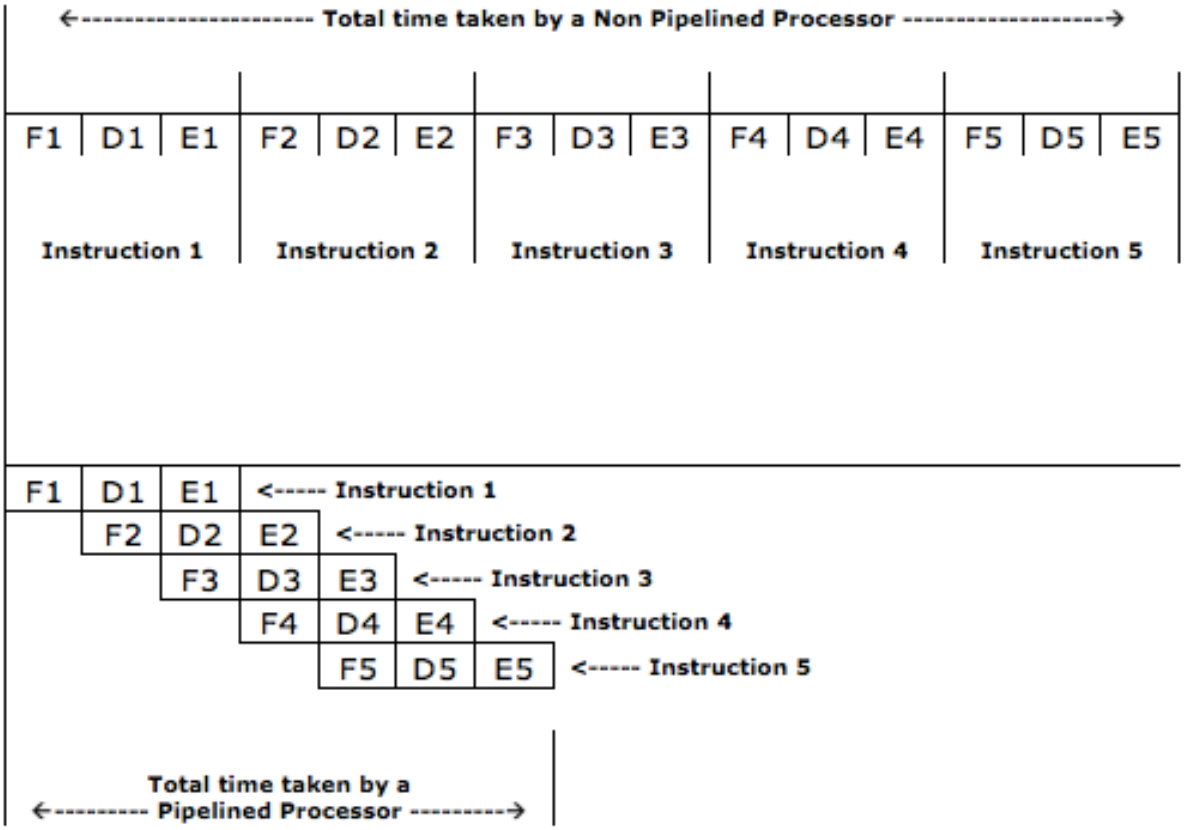


PIPELINED PROCESSOR EG: 8086



3 STAGE PIPELINING – 80386/ ARM 7

Here the instruction process in divided into three stages of **fetching, decoding and execution** and are overlapped. Hence **three instructions** are being processed at any point of time.


Total time taken by a Non Pipelined Processor
F1 D1 E1 F2 D2 E2 F3 D3 E3 F4 D4 E4 F5 D5 E5
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
F1 D1 E1 <----- Instruction 1
F2 D2 E2 <----- Instruction 2
F3 D3 E3 <----- Instruction 3
F4 D4 E4 <----- Instruction 4
F5 D5 E5 <----- Instruction 5
Total time taken by a Pipelined Processor

4 STAGE PIPELINING

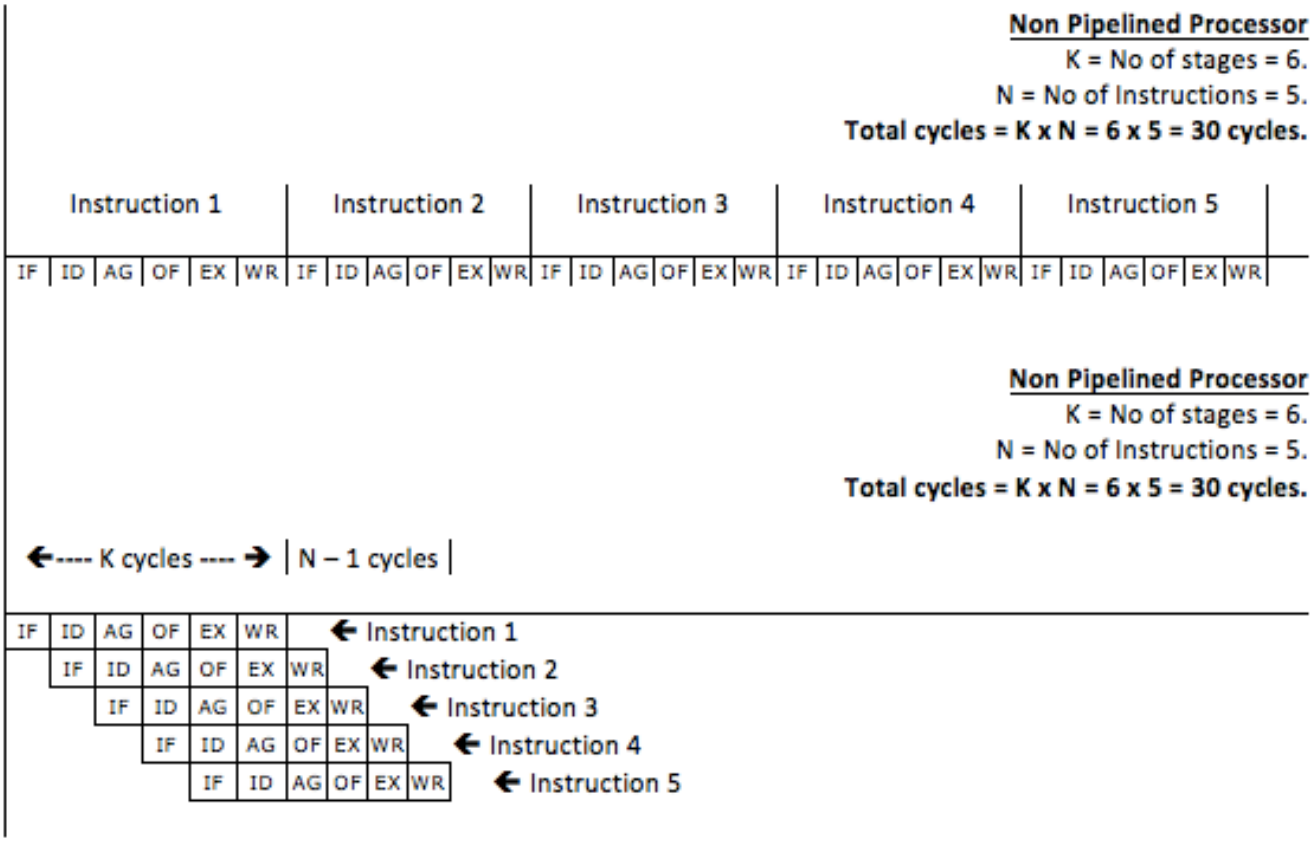
Fetch, Decode, Execute, Store

5 STAGE PIPELINING - PENTIUM

Fetch, Decode, Address Generation, Execute, Store

6 STAGE PIPELINING – PENTIUM PRO

Instruction Fetch (IF):	Fetch the instruction
Instruction Decode (ID):	Decode the instruction.
Address Generation (AG):	Calculate address of Memory operand
Operand Fetch (OF):	Fetch memory operand
Execute (EX):	Execute the operation
Write Back (WR):	Write back/ Store the result


Non Pipelined Processor
K = No of stages = 6.
N = No of Instructions = 5.
Total cycles = K x N = 6 x 5 = 30 cycles.
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
IF ID AG OF EX WR
Non Pipelined Processor
K = No of stages = 6.
N = No of Instructions = 5.
Total cycles = K x N = 6 x 5 = 30 cycles.
K cycles
N – 1 cycles
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5

Modern processors have a very deep pipeline.
Ex: Pentium 4 (Net burst Architecture) has a 20 stage pipeline.

ADVANTAGE OF PIPELINING

The obvious advantage of pipelining is that it increases the performance. As shown by the various examples above, deeper the pipelining, more is the level of parallelism, and hence the processor becomes much faster.

DRAWBACKS/ HAZARDS OF PIPELINING

There are various hazards of pipelining, which **cause a dip** in the performance of the processor. These hazards become even **more prominent** as the **number of pipeline stages increase**. They may occur due to the following reasons.

1) DATA HAZARD/ DATA DEPENDENCY HAZARD

Data Hazard is caused when **the result (destination) of one instruction becomes the operand (source) of the next instruction**.

Consider two instructions I1 and I2 (I1 being the first).

Assume I1: INC [4000H]
Assume I2: MOV BL , [4000H]

Clearly in I2, BL should get the incremented value of location [4000H].
But this can only happen once I1 has completely finished execution and also written back the result at [4000H].
In a multistage pipeline, I2 may reach execution stage before I1 has finished storing the result at location [4000H], and hence get a wrong value of data.
This is called **data dependency hazard**.
It is solved by inserting NOP (No operation) instructions between such data dependent instructions.

2) CONTROL HAZARD/ CODE HAZARD

Pipelining assumes that the program will always flow in a sequential manner.
Hence, it performs various stages of the forthcoming instructions before-hand, while the current instruction is still being executed.
While programs are sequential most of the times, it is not true always.
Sometimes, branches do occur in programs.
In such an event, all the forthcoming instructions that have been fetched/ decoded etc have to be flushed/ discarded, and the process has to start all over again, from the branch address. This causes pipeline bubbles, which simply means time of the processor is wasted.

Consider the following set of instructions:

```
Start:
        JMP Down
        INC BL
        MOV CL, DL
        ADD AL, BL
        …
        …
        …

Down:   DEC CH
```

JMP Down is a branch instruction.
After this instruction, program should jump to the location "Down" and continue with DEC CH instruction.

But, in a multistage pipeline processor, the sequentially next instructions after JMP Down have already been fetched and decoded. These instructions will now have to be discarded and fetching will begin all over again from DEC CH. This will keep several units of the architecture idle for some time. This is called a pipeline bubble.

The **problem of branching is solved** in higher processors by a method called "**Branch Prediction Algorithm**". It was introduced by **Pentium** processor. It relies on the **previous history** of the instruction as most programs are repetitive in nature. It then **makes a prediction** whether branch will be **taken or not** and hence puts the correct instructions in the pipelines.

3) STRUCTURAL HAZARD

Structural hazards are caused by **physical constraints in the architecture like the buses**. Even in the most basic form of pipelining, we want to execute one instruction and fetch the next one. Now as long as execution only involves registers, pipelining is possible. But **if execution requires to read/ write data from the memory, then it will make use of the buses, which means fetching cannot take place at the same time**. So the fetching unit will have to wait and hence a pipeline bubble is caused. This problem is solved in complex Harvard architecture processors, which use separate memories and separate buses for programs and data. This means fetching and execution can actually happen at the same time without any interference with each other.
E.g.: PIC 18 Microcontroller.

ADDRESSING MODES OF 8086 (ADDRESSING MODES OF IA 32 ARCHITECTURE)

Addressing modes is the manner in which operands are given in an instruction.
Most processors have various addressing modes, used in their instruction set.
The addressing modes of 8086 are as follows.

I IMMEDIATE ADDRESSING MODE

In this mode the **operand** is specified in the **instruction** itself.
Instructions are **longer** but the **operands** are **easily identified**.

```
Eg:   MOV CL, 12H        ; Moves 12 immediately into CL register
      MOV BX, 1234H      ; Moves 1234 immediately into BX register
```

II REGISTER ADDRESSING MODE

In this mode **operands** are specified using **registers**.
Instructions are **shorter** but **operands cant** be **identified** by looking at the instruction.

```
Eg:   MOV CL, DL         ; Moves data of DL register into CL register
      MOV AX, BX         ; Moves data of BX register into AX register
```

III DIRECT ADDRESSING MODE

In this mode **address** of the operand is directly specified **in the instruction**.

```
Eg:   MOV CL, [2000H]    ; CL Register gets data from memory location 2000H
                         ; CL ← [2000H]
Eg:   MOV [3000H], DL    ; Memory location 3000H gets data from DL Register
                         ; [3000H] ← DL
```

IV INDIRECT ADDRESSING MODE

In Indirect Addressing modes, **address is given by a register.**
The register can be incremented in a **loop** to access a **series of locations.**
There are various sub-types of Indirect addressing mode.

REGISTER INDIRECT ADDRESSING MODE

This is the most basic form of indirect addressing mode.
Here address is simply **given by a register**.

```
Eg:   MOV CL, [BX]       ; CL gets data from a memory location pointed by BX
                         ; CL ← [BX]. If BX = 2000H, CL ← [2000H]
Eg:   MOV [BX], CL       ; CL is stored at a memory location pointed by BX
                         ; [BX] ← CL. If BX = 2000H, [2000H] ← CL.
```

REGISTER RELATIVE ADDRESSING MODE

Here address is given by a **register plus a numeric displacement**.

```
Eg:   MOV CL, [BX + 03H]  ; CL gets data from a location BX + 03H
                          ; CL ← [BX+03H]. If BX = 2000H, then CL ← [2003H]
Eg:   MOV [BX + 03H], CL  ; CL is stored at location BX + 03H
                          ; [BX+03H] ← CL. If BX = 2000H, then [2003H] ← CL.
```

BASE INDEXED ADDRESSING MODE

Here address is given by **a sum of two registers**.
This is typically useful in accessing an array or a look up table.
One register acts as the base of the array holding its starting address and the **other acts as an index** indicating the element to be accessed.

```
Eg:   MOV CL, [BX + SI]   ; CL gets data from a location BX + SI
                          ; CL ← [BX+SI].
                          ; If BX = 2000H, SI = 1000H, then CL ← [3000H]

Eg:   MOV [BX + SI], CL   ; CL is stored at location BX + SI
                          ; [BX+SI] ← CL.
                          ; If BX = 2000H, SI = 1000H, then [3000H] ← CL.
```

BASE RELATIVE PLUS INDEX ADDRESSING MODE

Here address is given by a sum of base register plus index register plus a numeric displacement.

```
Eg: MOV CL, [BX+SI+03H]   ; CL gets data from a location BX + SI + 03H
                          ; CL ← [BX+SI+03H].
                          ; If BX = 2000H, SI = 1000H, then CL ← [3003H]

Eg: MOV [BX+SI+03H], CL   ; CL is stored at location BX + SI + 03H
                          ; [BX+SI+03H] ← CL.
                          ; If BX = 2000H, SI = 1000H, then [3003H] ← CL.
```

V IMPLIED ADDRESSING MODE

In this addressing mode, the operand is not specified at all, as it is an implied operand. Some instructions operate only on a particular register. In such cases, specifying the register becomes unnecessary as it becomes implied.

```
Eg:   STC                 ; Sets the Carry flag.
                          ; This instruction can only operate on the Carry Flag.
Eg:   CMC                 ; Complements the Carry flag.
                          ; This instruction can only operate on the Carry Flag.
```

ADDRESSING MODES OF 8085

Addressing modes is the manner in which operands are given in an instruction.
Most processors have various addressing modes, used in their instruction set.
The addressing modes of 8085 are as follows.

IMMEDIATE ADDRESSING MODE

In this mode, the **Data** is specified **in** the **Instruction** itself.

Eg: **MVI A, 35H** *; Move immediately the value 35 into the Accumulator. i.e. A ← 35H*
LXI B, 4000H *; Move immediately the value 4000 into BC register pair. i.e. BC ← 4000H*

Advantage:
Programmer can easily **identify** the **operands**.

Disadvantage:
Always more than one byte hence requires **more space**.
The µP requires **two or three machine cycles** to fetch the instruction hence **slow**.

REGISTER ADDRESSING MODE

In this mode, the **Data** is specified **in Registers**.

Eg: **MOV B, C** *; Move the Contents of C-Register into B-Register. i.e. B ← C*
INR B *; Increments the contents of B-Register. i.e. B ← B + 1*

Advantage:
Instructions are of **one byte** so only one cycle is required to fetch them.

Disadvantage:
Operands **cannot** be easily **identified**.

DIRECT ADDRESSING MODE

In this mode, the **Address** of the operand is specified **in** the **Instruction** itself.

Eg: **LDA 2000H** *; Loads "A" register with Contents of Location 2000. i.e. A ← [2000]*
STA 2000H *; Stores the Contents of "A" register at the Location 2000. i.e. [2000] ← A*

Advantage:
The programmer **can identify** the address of the operand.

Disadvantage:
These are **three byte instructions** hence require three fetch cycles.

INDIRECT ADDRESSING MODE

In this mode, the **Address** of the operand is specified **in Registers.**
Hence, the instruction indirectly points to the operands.

E.g.:: **LDAX B** *; Loads "A" register with the contents of the memory Location pointed by BC Pair*
; So if BC pair = 4000 i.e. [BC] = 4000 then A ← [4000].
#Please refer Bharat Sir's Lecture Notes for this ...

STAX B *; Stores the contents of the Accumulator at the memory location pointed by BC pair.*
; So if contents of BC pair = 4000 i.e. [BC] = 4000 then [4000] ← A.
#Please refer Bharat Sir's Lecture Notes for this ...

Advantage:
Address of the operand is **not fixed** and hence can be used in a **loop**.

Disadvantage:
Requires initialization of the register pair hence more **complex**.

IMPLIED ADDRESSING MODE

In this mode, the **Operand** is **implied** in the instruction.
This instruction will work only on that implied operand, and not on any other operand.

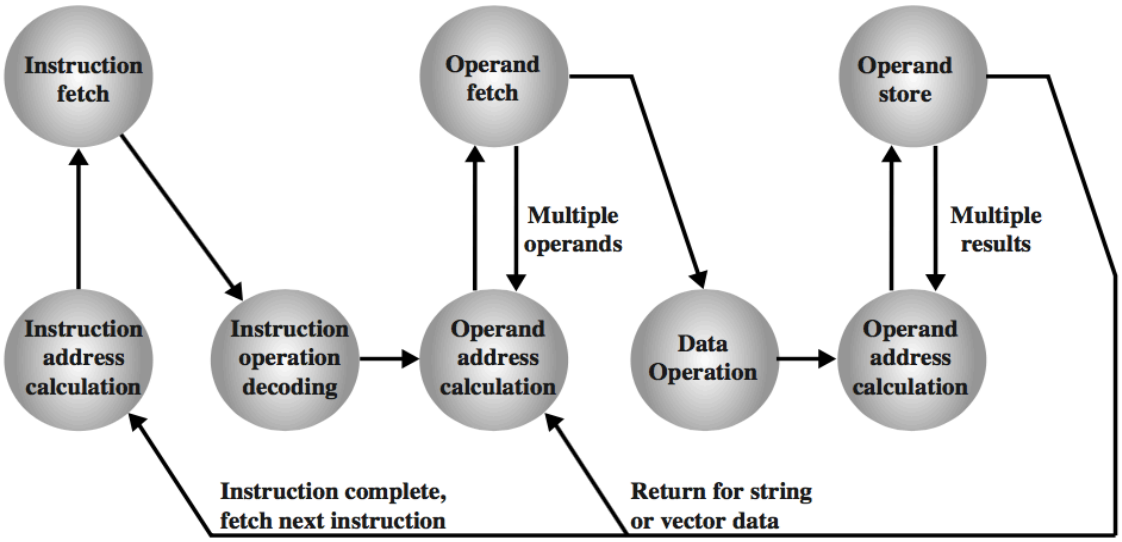
Eg: **STC** *; Sets the Carry Flag in the Flag register, Cy ← 1.*
CMC *; Complements the Carry Flag in the Flag register.*

Advantage:
Instructions are generally **only one byte**.

Disadvantage:
Rigid, as it works only on a fixed operand.

INSTRUCTION CYCLE

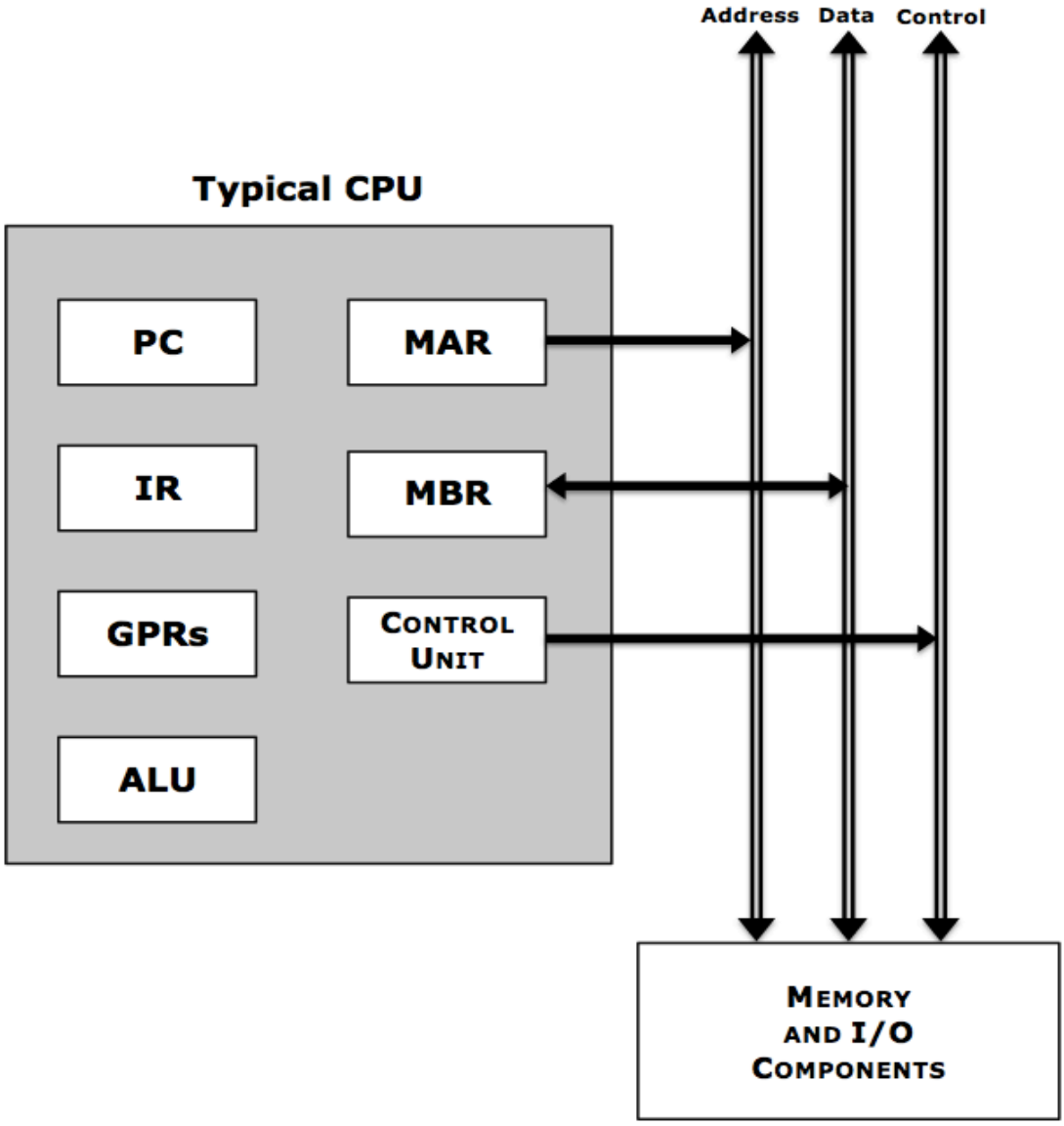
An instruction cycle is the complete process of fetching, decoding and executing the instruction.



1) **PC gives the address to fetch** an instruction from the memory.
2) Once fetched, the **instruction opcode is decoded**.
3) This **identifies**, if there are **any operands to be fetched** from the memory.
4) The **operand address is calculated.**
5) **Operands are fetched** from the memory.
6) Now the **data operation is performed** on the operands, and a **result is generated**.
7) If the result has to be stored in a **register**, the **instruction ends** here.
8) If the **destination is memory**, then first the **destination address has to be calculated**.
9) The result is then **stored in the memory**.
10) Now the current instruction has **been executed**.
11) **Side by side PC is incremented** to calculate address of the next instruction.
12) The above instruction cycle then **repeats for further instructions**.

MICRO-OPERATIONS & CONTROL SIGNALS

- A **Program** is a **set of Instructions**.
- An **Instruction**, requires a **set of small** operations called **Micro-Operations**.
- A **Micro-Operation** is a **finite activity performed** by the processor **in one clock cycle**. One clock cycle is also called one **T-state** (Transition state).
- **One Micro-Operation** requires **One T-state**.
- Several **Independent** Micro-Operations can be performed in the same T-state.
- **Control unit generates control signals** to perform these very Micro-Operations.
- To understand Control Units, we must first clearly understand **Micro-Operations**.



MICRO-OPERATIONS FOR INSTRUCTION FETCHING

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction from MBR*
	PC ← PC + 1	*PC gets incremented*

As we can see in the above table, two Micro-Operations can take place in the same T-state i.e.

IR← MBR and PC← PC + 1

This is because they are completely independent of each other.
In fact, PC becoming PC + 1 can also be performed in the 2nd T-state while the instruction is being fetched from the memory through the data bus into MBR.
This is shown below.

MICRO-OPERATIONS FOR INSTRUCTION FETCHING (ALTERNATE METHOD)

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
	PC ← PC + 1	*PC gets incremented*
T3:	IR ← MBR	*IR gets instruction from MBR*

Please Note

PC ← PC + 1 cannot take place in the 1st T-state.
That's because, in the 1st T-state, PC is providing the address on the address bus, through MAR. If at the same time PC gets incremented, then the incremented address will be put on the address bus.

Now that we know how an instruction is fetched, we can proceed further and learn Micro-Operations for various instructions, of different Addressing Modes.

MICRO-OPERATIONS FOR IMMEDIATE ADDRESSING MODE

E.g.: MOV R1, 25H; R1 register gets the immediate value 25H

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, 25H" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← 25H (IR)	*R1 register gets the value 25H from IR*

MICRO-OPERATIONS FOR REGISTER ADDRESSING MODE

E.g.: MOV R1, R2; R1 register gets the data from Register R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, R2" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	R1 ← R2	*R1 register gets the value from R2 Register*

MICRO-OPERATIONS FOR DIRECT ADDRESSING MODE

E.g.: MOV R1, [2000H]; R1 register gets the data from memory location 2000H

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [2000H]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← IR (2000H)	*MAR gets the address 2000H from IR*
T5:	MBR ← Memory ([2000H])	*MBR gets contents of location 2000H from Memory.*
T6:	R1 ← MBR ([2000H])	*Register R1 gets contents of memory location 2000H from MBR*

MICRO-OPERATIONS FOR INDIRECT ADDRESSING MODE

E.g.: MOV R1, [R2]; R1 register gets the data from memory location pointed by R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV R1, [R2]" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← Memory ([R2])	*MBR gets contents of location pointed by R2 from Memory.*
T6:	R1 ← MBR ([R2])	*Register R1 gets contents of memory location pointed by R2 from MBR*
		In the exam, once, Add R1, [R2] was asked. *Everything else will be same. Only change: T6: R1 ← R1 + MBR*

MICRO-OPERATIONS FOR INDIRECT ADDRESSING MODE

E.g.: MOV [R2], R1; R1 register stores data into memory location pointed by R2

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "MOV [R2], R1" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	MAR ← R2	*MAR gets the address R2 Register*
T5:	MBR ← R1	*R1 puts data into MBR to store it in the memory location pointed by R2.*

MICRO-OPERATIONS FOR IMPLIED ADDRESSING MODE

E.g.: STC; Set the Carry Flag; (CF ← 1).

STATE	MICRO-OPERATION	EXPLANATION
T1:	MAR ← PC	*PC puts address of the next instruction into MAR*
T2:	MBR ← Memory (Instr)	*MBR gets instruction from memory through data bus*
T3:	IR ← MBR	*IR gets instruction "STC" from MBR*
	PC ← PC + 1	*PC gets incremented*
T4:	CF ← 1	*Carry Flag in the Flag Register becomes 1*

HARDWIRED CONTROL UNIT

- Here control signals are produced by hardware.
- There are three types of Hardwired Control Units

STATE TABLE METHOD

1) It is the most basic type of hardwired control unit.
2) Here the behavior of the control unit is represented in the form of a table called the state table.
3) The rows represent the T-states and the columns indicate the instructions.
4) Each intersection indicates the control signal to be produced, in the corresponding T-state of every instruction.
5) A circuit is then constructed based on every column of this table, for each instruction.

T-STATES	INSTRUCTIONS			
	I_1	I_2	…	I_N
T_1	$Z_{1,1}$	$Z_{1,2}$	…	$Z_{1,N}$
T_2	$Z_{2,1}$	$Z_{2,2}$	…	$Z_{2,N}$
…	…	…	…	…
T_M	$Z_{M,1}$	$Z_{M,2}$	…	$Z_{M,N}$

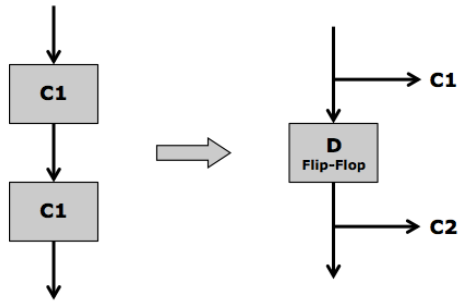
$Z_{1,1}$: Control Signal to be produced in T-state (T_1) of Instruction (I_1)

ADVANTAGE:
It is the simplest method and is ideally suited for very small instruction sets.

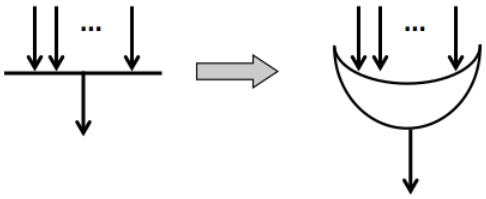
DRAWBACK:
As the number of instructions increase, the circuit becomes bigger and hence more complicated.
As a tabular approach is used, instead of a logical approach (flowchart), there are duplications of many circuit elements in various instructions.

DELAY ELEMENT METHOD

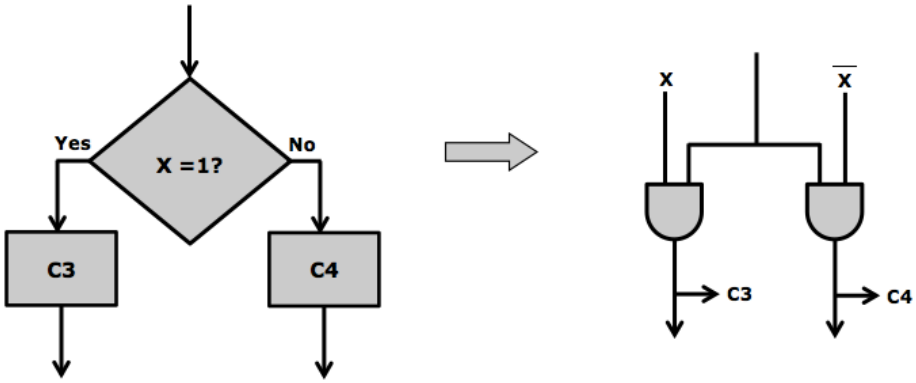
1) Here the behavior of the control unit is represented in the form of a flowchart.
2) Each step in the flowchart represents a control signal to be produced.
3) Once all steps of a particular instruction, are performed, the complete instruction gets executed.
4) Control signals perform Micro-Operations, which require one T-states each.
5) Hence between every two steps of the flowchart, there must be a delay element.
6) The delay must be exactly of one T-state. This delay is achieved by D Flip-Flops.
7) These **D Flip-Flops** are **inserted** between every two **consecutive control signals**.



8) Of all D Flip-Flops only one will be active at a time. So the method is also called **"One Hot Method"**.
9) In a **multiple entry point**, to combine two or more paths, we use an **OR gate**.



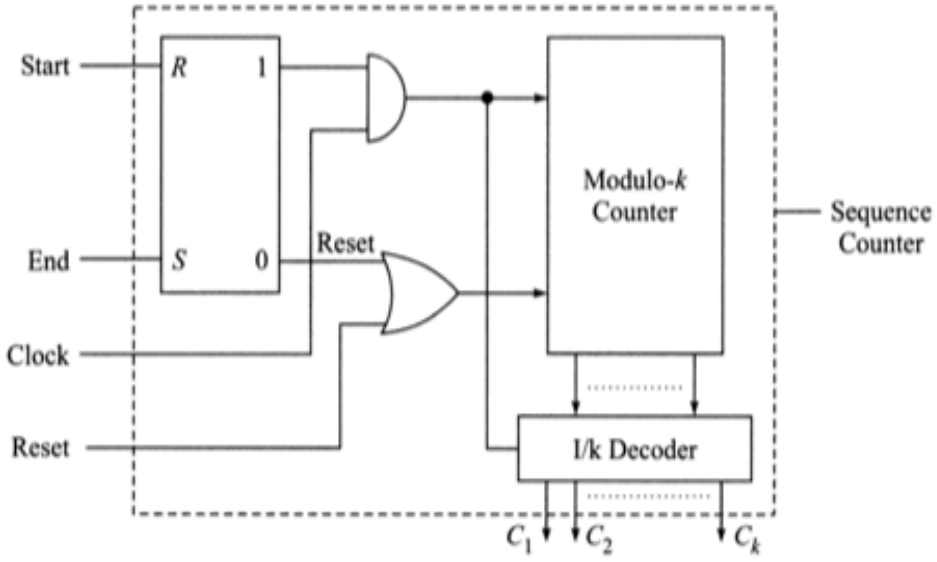
10) A **decision box** is replaced by a set of two complementing **AND gates**



ADVANTAGE:
As the method has a logical approach, it can **reduce the circuit complexity**.
This is done by re-utilizing common elements between various instructions.

DRAWBACK:
As the no of instructions increase, the number of **D Flip-Flops increase**, so the **cost increases**.
Moreover, **only one of those D Flip-Flops are actually active at a time.**

SEQUENCE COUNTER METHOD



1) This is the **most popular** form of hardwired control unit.
2) It follows the same **logical approach of a flowchart**, like the Delay element method, but does not use all those unnecessary D Flip-Flops.
3) First a flowchart is made representing the behavior of a control unit.
4) It is then **converted into a circuit** using the same principle of AND & OR gates.
5) We need a **delay of 1 T-state** (one clock cycle) between every two **consecutive control signals**.
6) That is achieved by the above circuit.
7) If there are **"k" number of distinct steps** producing control signals, we employ a **"mod k"** and **"k" output decoder**.
8) The counter will **start counting** at the beginning of the instruction.
9) The "clock" input via an AND gate ensures each count will be generated **after 1 T-state**.
10) The count is given to the **decoder** which **triggers the generation of "k" control signals**, each after a delay of 1 T-state.
11) When the **instruction ends**, the **counter is reset** so that next time, it begins from the first count.

ADVANTAGE:

Avoids the use of **too many D Flip-Flops**.

GENERAL DRAWBACKS OF A HARDWIRED CONTROL UNIT

1) Since they are based on hardware, as the instruction set increases, the **circuit becomes more and more complex**. For modern processors having hundreds of instructions, it is virtually impossible to create Hardwired Control Units.
2) Such large circuits are **very difficult to debug**.
3) As the **processor gets upgraded**, the entire Control Unit has to be **redesigned**, due to the **rigid nature** of hardware design.

MICROPROGRAMMED CONTROL UNIT

WILKES' DESIGN FOR A MICROPROGRAMMED CONTROL UNIT

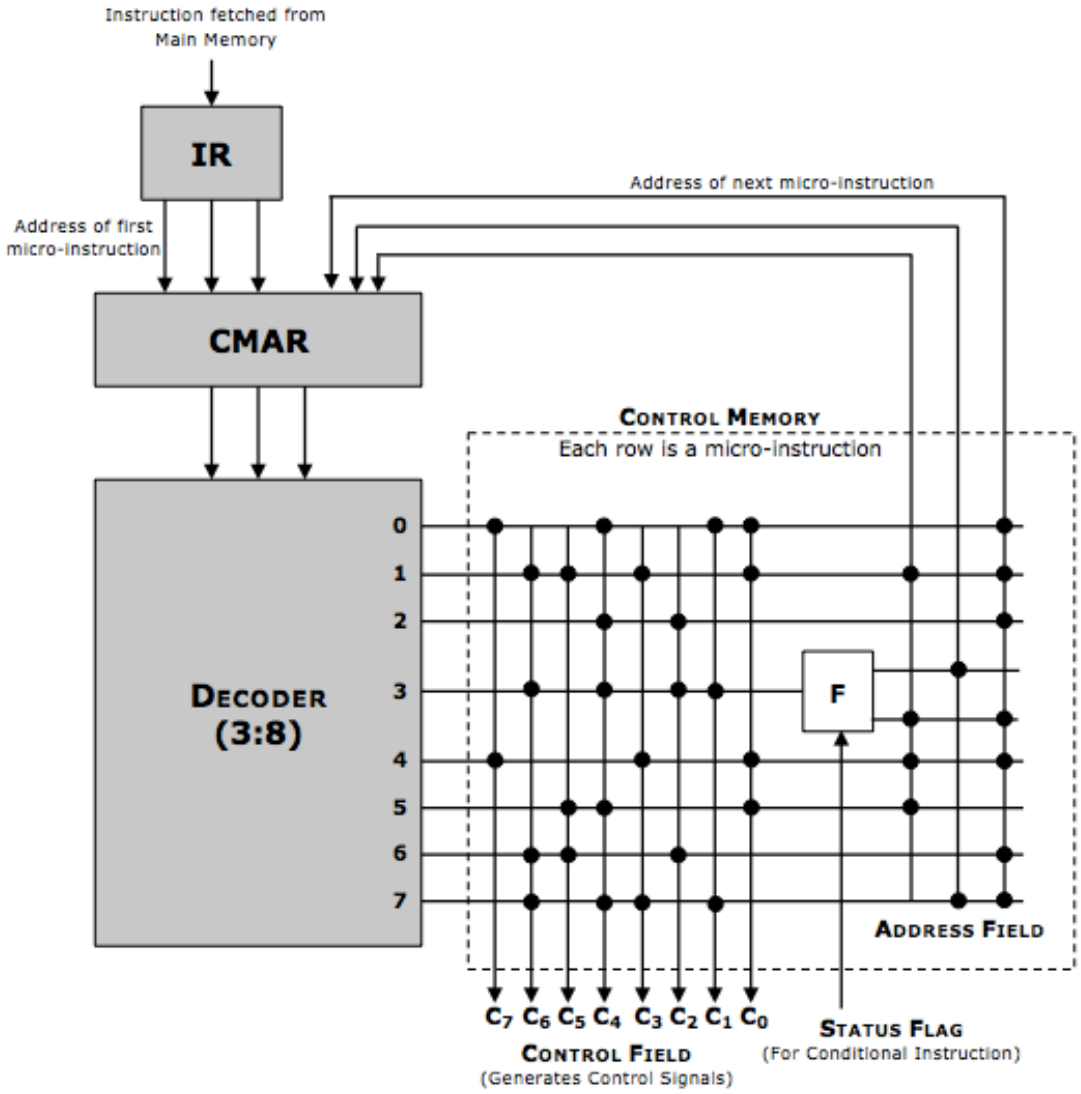
1) Microprogrammed Control Unit produces control signals by **software**, using **micro-instructions**
2) A program is a set of instructions.
3) An instruction requires a set of Micro-Operations.
4) **Micro-Operations are performed by control signals**.
5) Instead of generating these control signals by hardware, **we use micro-instructions**.
6) This means **every instruction requires a set of micro-instructions**
7) **This is called its micro-program**.
8) Microprograms for all instructions are **stored in a small memory called "Control Memory"**.
9) The Control memory is **present inside the processor**.
10) Consider an **Instruction** that is **fetched from the main memory** into the Instruction Register (**IR**).
11) The processor uses its unique **"opcode"** to identify the **address of the first micro-instruction**.
12) That address is loaded into **CMAR** (Control Memory Address Register).
13) CMAR passes the address to the **decoder**.
14) The decoder **identifies the corresponding micro-instruction** from the Control Memory.
15) A micro-instruction has **two fields**: a control filed and an address field.
16) **Control field:** Indicates the **control signals to be generated**.
17) **Address field:** Indicates the **address of the next micro-instruction**.
18) This address is further **loaded into CMAR** to **fetch the next** micro-instruction.
19) For a **conditional micro-instruction**, there are **two address fields**.
20) This is because, the address of the next micro-instruction **depends on the condition**.
21) The condition (true or false) is **decided by the appropriate control flag**.
22) The control memory is usually implemented using FLASH ROM as it is writable yet non volatile.

ADVANTAGES

1) The biggest advantage is **flexibility.**
2) Any **change** in the control unit can be performed by **simply changing the micro-instruction**.
3) This makes **modifications and up gradation** of the Control Unit **very easy**.
4) Moreover, software can be **much easily debugged** as compared to a large Hardwired Control Unit.

DRAWBACKS

1) **Control memory** has to be present **inside** the processor, **increasing its size.**
2) This also **increases the cost** of the processor.
3) The **address field** in every micro-instruction **adds more space** to the control memory. This can be easily **avoided** by proper **micro-instruction sequencing**.


Instruction fetched from Main Memory
IR
Address of next micro-instruction
Address of first micro-instruction
CMAR
CONTROL MEMORY
Each row is a micro-instruction
DECODER (3:8)
0
1
2
3
4
5
6
7
F
ADDRESS FIELD
C7 C6 C5 C4 C3 C2 C1 C0
CONTROL FIELD
(Generates Control Signals)
STATUS FLAG
(For Conditional Instruction)

TYPICAL MICROPROGRAMMED CONTROL UNIT *(Super Important!)*

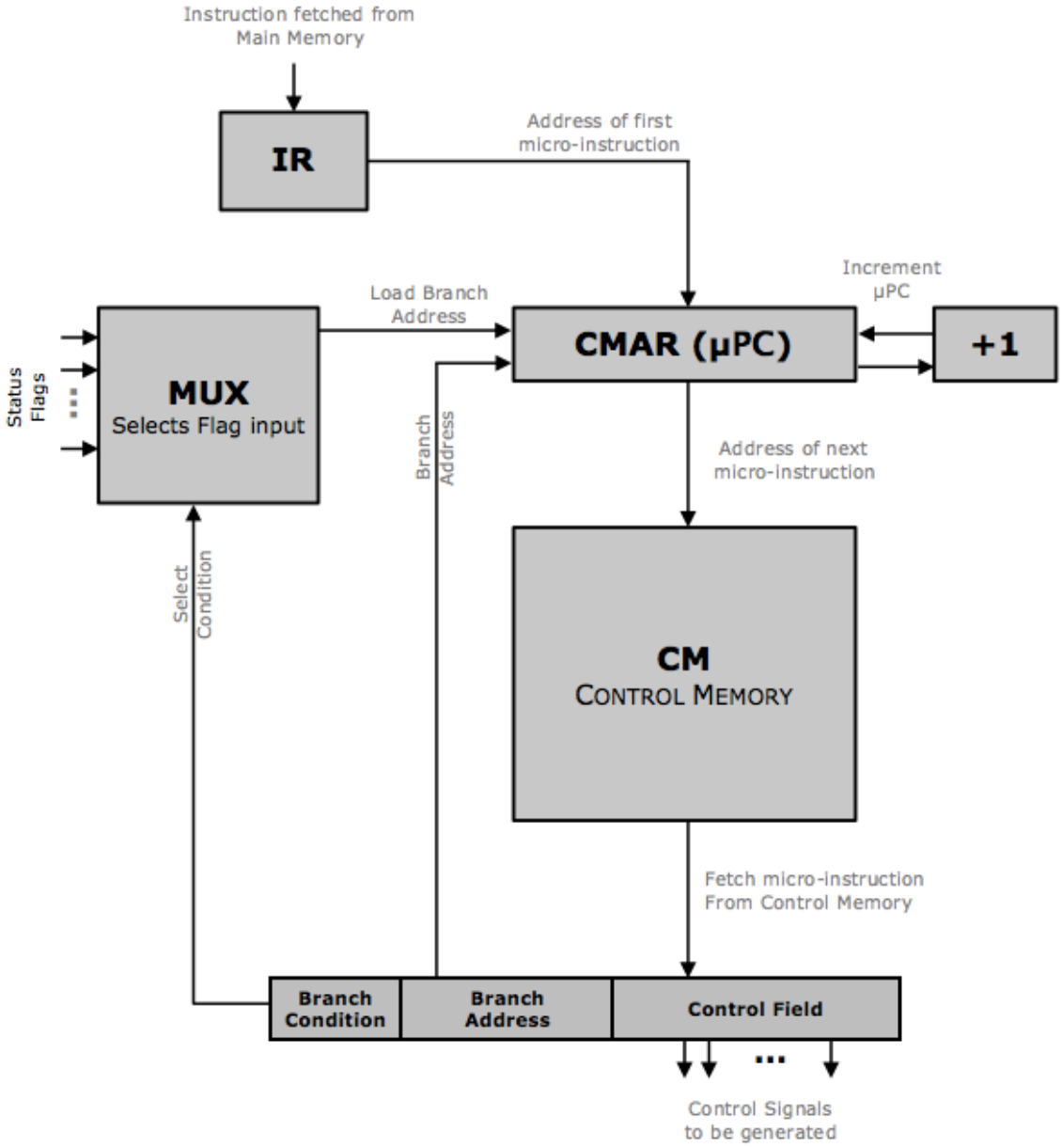
1) Microprogrammed Control Unit produces control signals by **software**, using **micro-instructions**.
2) A program is a set of instructions.
3) An instruction requires a set of Micro-Operations.
4) **Micro-Operations are performed by control signals**.
5) Here, these control signals are generated using **micro-instructions**.
6) This means **every instruction requires a set of micro-instructions**
7) **This is called its micro-program**.
8) Microprograms for all instructions are **stored in a small memory called "Control Memory"**.
9) The Control memory is **present inside the processor**.
10) Consider an **Instruction** that is **fetched from the main memory** into the Instruction Register (**IR**).
11) The processor uses its unique **"opcode"** to identify the **address of the first micro-instruction**.
12) That address is loaded into **CMAR** (Control Memory Address Register) also called **μIR**.
13) This address is **decoded** to **identify the corresponding μ-instruction** from the Control Memory.
14) There is a big **improvement over Wilkes' design**, to reduce the size of micro-instructions.
15) Most micro-instructions will **only have a Control field**.
16) The **Control field** Indicates the **control signals to be generated**.
17) Most micro-instructions **will not have an address field**.
18) Instead, **μPC will simply get incremented** after every micro-instruction.
19) This is as long as the μ-program is executed **sequentially**.
20) If there is a **branch μ-instruction** only then there will be an **address filed**.
21) If the branch is **unconditional**, the **branch address will be directly loaded into CMAR.**
22) For **Conditional branches**, the Branch condition will **check the appropriate flag**.
23) This is done using a **MUX** which has all flag inputs.
24) If the **condition is true**, then the **MUX will inform CMAR to load the branch address**.
25) If the **condition is false** CMAR will simply get **incremented**.
26) The control memory is usually implemented using **FLASH ROM** as it is **writable** yet **non volatile**.

ADVANTAGES

1) The biggest advantage is **flexibility.**
2) Any **change** in the control unit can be performed by **simply changing the micro-instruction**.
3) This makes **modifications and up gradation** of the Control Unit **very easy**.
4) Moreover, software can be **much easily debugged** as compared to a large Hardwired Control Unit.
5) Since most micro-instructions are executed sequentially, they don't **need for an address field**.
6) This significantly **reduces the size of micro-instructions**, and hence the **Control Memory**.

DRAWBACKS

1) **Control memory** has to be present **inside** the processor, **increasing its size.**
2) This also **increases the cost** of the processor.


Instruction fetched from Main Memory
IR
Address of first micro-instruction
Increment µPC
Load Branch Address
CMAR (µPC)
+1
Status Flags
MUX
Selects Flag input
Branch Address
Address of next micro-instruction
Select Condition
CM
CONTROL MEMORY
Fetch micro-instruction From Control Memory
Branch Condition
Branch Address
Control Field
Control Signals to be generated

APPLICATIONS OF MICROPROGRAMMED CONTROL UNIT / MICROPROGRAMMING

Microprogramming has various advantages like flexibility, simplicity, cost effectiveness etc.
As a result, it plays a major role in the following applications.

1) DEVELOPMENT OF CONTROL UNITS

Modern processors have very **large and complex instruction sets**.
Microprogramming is ideally suited for **making Control Units** of such processors as they are far less complex and can be **easily modified**.

2) USER TAILORING OF THE CONTROL UNIT

As the Control Unit is developed using software, it can be **easily reprogrammed**.
This can be used for **custom made modifications** of the Control Unit.
To allow this, the Control Memory must be **writeable like RAM or Flash ROMs**.

3) EMULATION

Emulation is when one **processor (A) is made to emulate or behave like another processor (B)**.
To do this, "A" must be able to execute the instructions of "B".
If we re-program the Control Memory of "A", same as that of "B", then "A" will be able to emulate the behavior of "B", for every instruction. This is possible only in Microprogrammed Control Units.
Used generally when a **main processor** has to **emulate** the behavior of a **math co-processor**.

4) IMPROVING THE OPERATING SYSTEM

Microprogramming can be used to implement **complex and secure functions of the OS**.
This not only makes the OS **more powerful and efficient**, but more importantly **secure**, as it provides the OS a higher degree of protection from malicious **virus attacks**.

5) HIGH LEVEL LANGUAGE SUPPORT

Modern high level languages have **far more advanced and complex data types**.
Microprogramming can **provide support for such data types** directly from the processor level.
As a result the language becomes **easy to compile and also faster to execute**.

6) MICRO-DIAGNOSTICS

As Microprogrammed Control Units are software based, **debugging an error is far more easy** as compared to doing the same for a complex hardwired control unit. This allows **monitoring, detection, isolation and repairs** of any kind of **system errors** in the Control Unit.
At times, it can also act as a **runtime substitute**, if the corresponding **hardwired component fails**.

7) DEVELOPMENT OF SPECIALIZED PROCESSORS

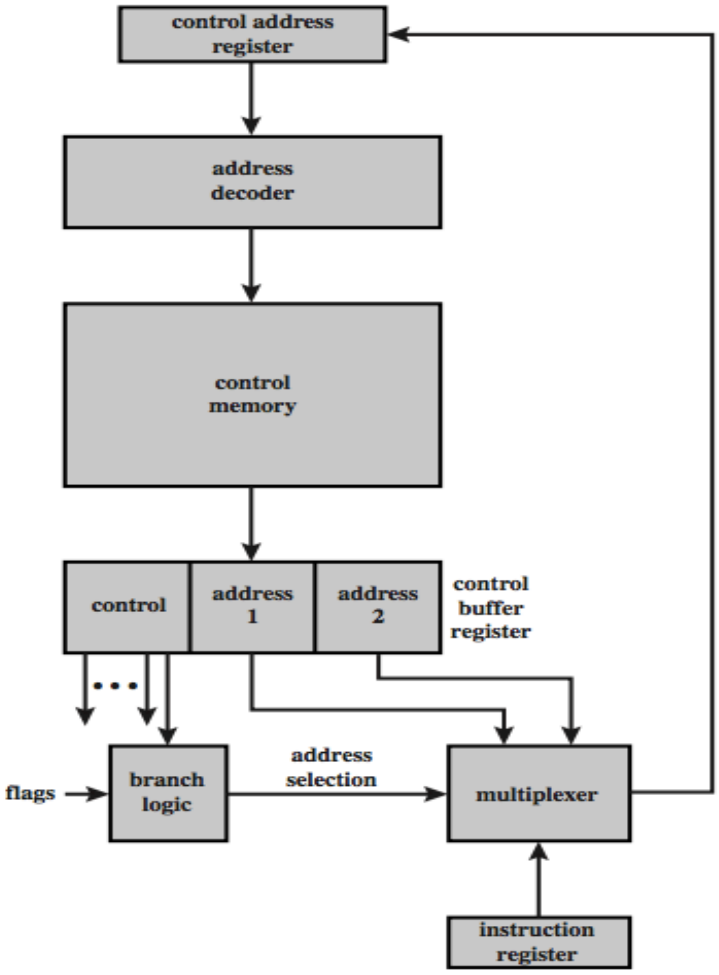
All processors are not general purpose. Many applications require **specialized processors** like **DSP** (Digital Signal Processors) for **communication**, **GPU** (Graphic Processor Unit) for **image processing**.
They have **complex instruction sets** and also need to be **constantly upgraded**, making Microprogrammed Control Unit the **ideal choice**.

MICRO-INSTRUCTION SEQUENCING

It is the method of determining the flow of the Microprogram.
There are two main techniques.

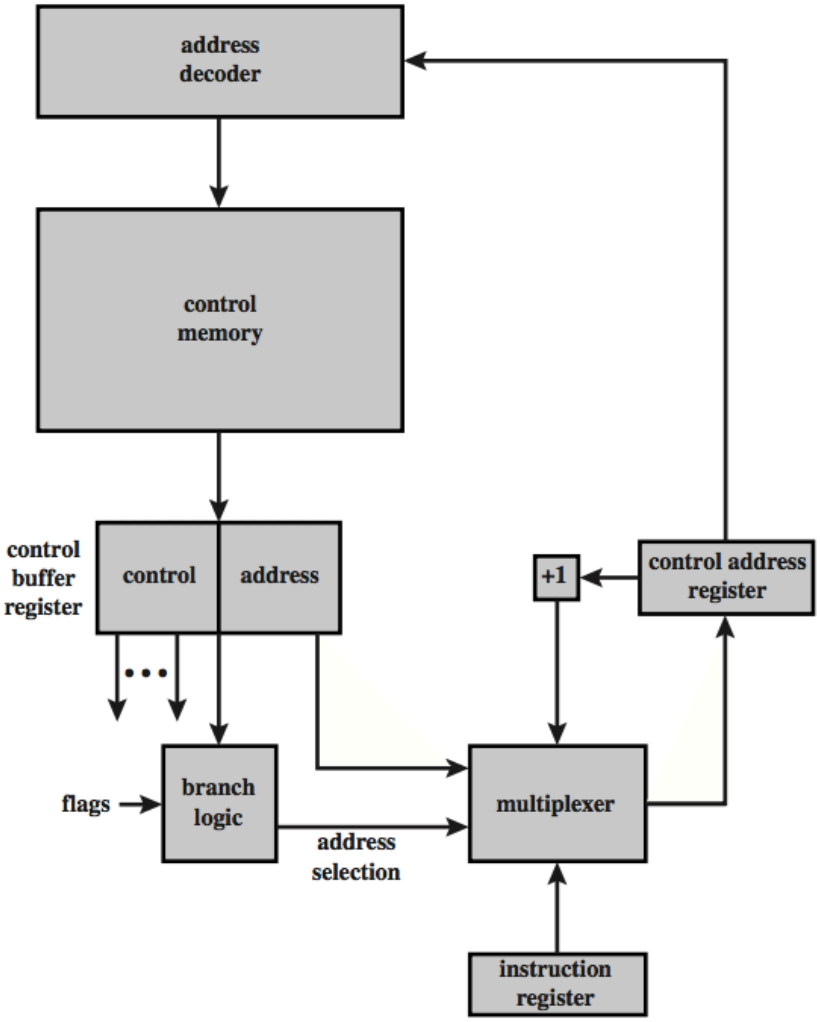
1) DUAL ADDRESS FIELD

In this approach, micro-instructions are **NOT executed in a sequential manner**.
The instruction register (**IR**) gives the address of the **first micro-instruction**.
Thereafter, **each micro-instruction gives the address of the next micro-instruction**.
If it is a **conditional micro-instruction**, it will contain **two address fields**.
One for the condition to be **true** and the other for **false**.
Hence it is called **dual address field**.
The **multiplexer will decide the address** the address to be loaded into the control memory address register (CMAR) **based on the status flags**.



2) SINGLE ADDRESS FIELD

In this approach, micro-instructions **are executed in a sequential manner**.
The instruction register (**IR**) gives the address of the **first micro-instruction** into CMAR.
Thereafter the **address is simply incremented**.
Hence every micro-instruction **need not carry address** of the next one.
This is true so long as the μ-program is executed in a **sequential manner**.
For an **unconditional branch**, the micro-instruction **contains the branch address**.
This address will be **loaded into CMAR**.
For a **conditional branch**, the micro-instruction contains the **branch address for true condition**.
If the **condition is false**, the current address in **CMAR will be simply incremented**.
This means even in the **worst case**, the micro-instruction will **carry only one address**.
Hence it is called **single address field**.
The **multiplexer will decide the address** the address to be loaded into the control memory address register (CMAR) **based on the status flags**.

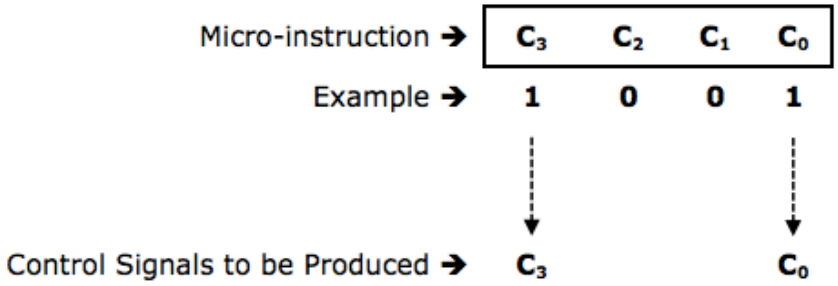


MICRO-INSTRUCTION FORMAT

The main part of the micro-instruction is its control field.
It determines the control signals to be produced.
It can be of two different formats: Horizontal or Vertical.

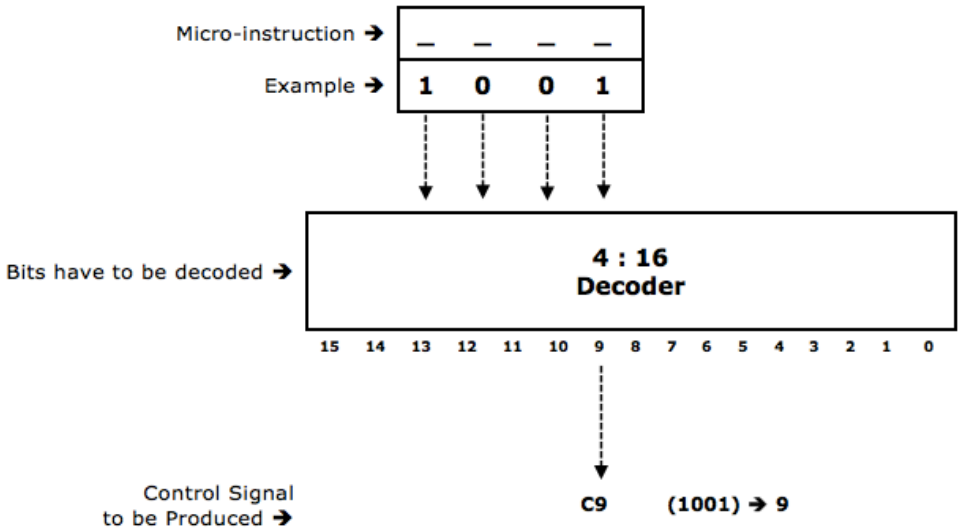
1) HORIZONTAL MICRO-INSTRUCTION

Here every bit of the micro-instruction corresponds to a control signal.
Whichever bit is "1", that particular control signal will be produced by the micro-instruction.



2) VERTICAL MICRO-INSTRUCTION

Here bits of the micro-instruction have to be decoded.
The decoded output decides the control signal to be produced.

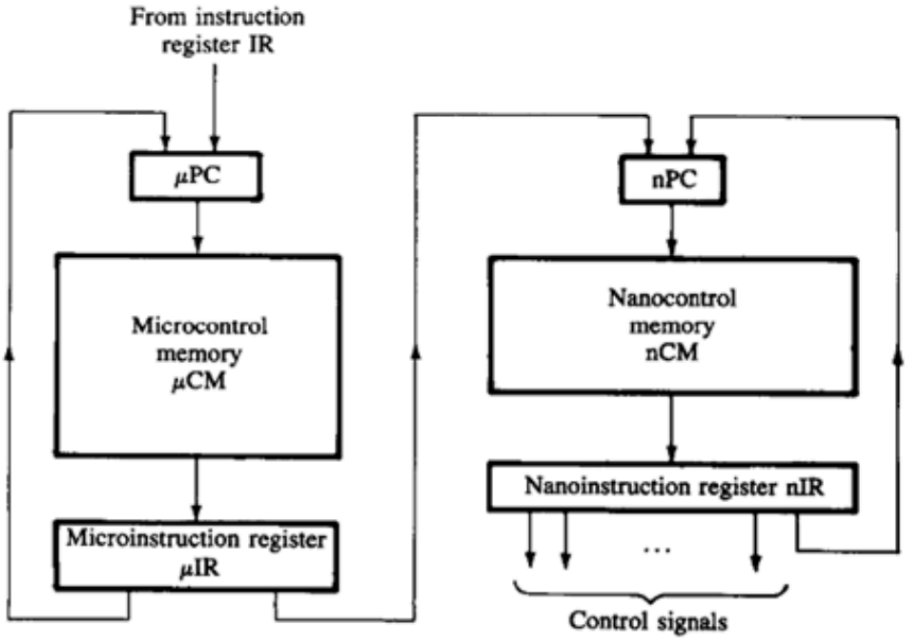

Micro-instruction →
_ _ _ _
Example →
1 0 0 1
Bits have to be decoded →
4 : 16
Decoder
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Control Signal
to be Produced →
C9
(1001) → 9

	HORIZONTAL MICRO-INSTRUCTION	VERTICAL MICRO-INSTRUCTION
1	Every bit of the micro-instruction corresponds to a control signal.	Bits of the micro-instruction have to be decoded to produce control signals.
2	Does not require a decoder.	Needs a decoder.
3	N bits in the micro-instruction will totally produce N control signals.	N bits in the micro-instruction will totally produce 2^N control signals.
4	Multiple control signals can be produced by one micro-instruction.	Only one control signal can be produced by one micro-instruction.
5	As the control signals increase, the micro-instruction grows wider. Hence the Control Memory grows Horizontally.	To produce more control signals, more number of micro-instructions are needed. Hence the Control Memory grows Vertically.
6	Executes faster as no decoding needed.	Executes slower as decoding is needed.
7	Micro-instruction are very wide. Hence Control memory is large.	Micro-instruction are much narrower. Hence Control memory is small.
8	Circuit is simpler as a decoder is not needed.	Circuit is more complex as a decoder is needed.

As seen from the above comparison, both methods have their pros and cons.
So a combination of both is used together called Nano-Programming.

NANO-PROGRAMMING *(Very Important)*

1) **Horizontal** µ-instructions can produce **multiple control signals** simultaneously, but are **very wide**.
2) This makes the Control Memory **very large in size**.
3) **Vertical** micro-instructions are **narrow**, **but on decoding** can **produce** only **one control signal**.
4) This makes the Control Memory **small** but the execution is **slow**.
5) Hence a **combination of both techniques** is needed called **Nano-Programming**.
6) Here we have a **two level control memory**.
7) The instruction is fetched from the **main memory into IR**.
8) Using its **opcode** we load **address of its first micro-instruction** into **µPC**,
9) Using this address we **fetch the micro-instruction** from µ-Control Memory (**µCM**) **into µIR**.
10) This is in **vertical form** and has to be decoded.
11) The decoded output **loads a new address** in a Nano program counter (**nPC**).
12) Using this address we **fetch the Nano-instruction** from Nano-Control Memory (**nCM**) into **nIR**.
13) This is in **horizontal form** and can **directly generate control signals**.
14) Such a combination **gives advantage of both techniques**.
15) The size of the Control Memory is **small** as µ-instructions are **Vertical**.
16) Multiple control signals can be **produced simultaneously** as Nano-instructions are **Horizontal**.



	HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
1	**Control signals are generated using hardware**.	**Control signals are generated using software (Microprogram).**
2	Since **hardware** is used, the circuit is **rigid**.	Since **software** is used, the circuit is **flexible.**
3	**Modification** to the Control Unit requires **re-design** of the entire **hardware**.	**Modification** to the Control Unit simply requires **re-programming** of μ-instructions.
4	**Ideally suited** for processors with **small and simple instruction sets**.	**Ideally suited** for processors with **large and complex instruction sets**.
5	**Debugging** a large Hardwired Control Unit is **very difficult.**	As micro-programs are software, **debugging is much easier.**
6	**Emulation is not possible.**	**Emulation is possible.**
7	Executes **faster** as control signals are directly generated by hardware.	Executes **slower** as **time** is **wasted** in **fetching and decoding μ-instructions**.
8	Does not need a **Control Memory**.	Needs a **Control Memory**.
9	**Cost is lower** as Control Memory is not needed.	**Cost is higher** as Control Memory is needed inside the processor.
10	Preferred in **RISC processors**.	Preferred in **CISC processors**.

As seen from the above comparison, both methods have their pros and cons.
Hence modern processors use a combination of both.
Simple and regularly used instructions are decoded by a Hardwired Control Unit as they are faster.
Complex instructions are decoded by a Microprogrammed Control Unit as they are easier to design..

7) CACHE MAPPING TECHNIQUES

Blocks are loaded from Main Memory to Cache Memory.
Cache Mapping decides which block of Main Memory comes into which block of Cache Memory.
There are several mapping techniques trying to balance between Hit Ratio, Search-time and Tag size.
Every cache block has a **Tag** indicating which block of Main Memory is mapped into that block.
A collection of such tags is called the **cache director**y, very similar to a page table.
Cache directory is a part of the cache.
Since Cache Memory is **very expensive**, we need the cache directory to be as small as possible.
That means the **Tag must be of minimum size**.
The Main Memory address, issued by the processor contains the desired block number.
This is compared to the Tag of a Cache Block, which gives the Block number that is present.
If they are equal, **it's a Hit**. If Not, the search may have to be repeated for several other blocks.
It is obvious to understand, **the number of searches must be as low as possible.**
Finally, the Mapping technique must yield maximum utilization of the Cache memory space, so that the **Hit ratio remains as High as possible**.
There are three popular Cache Mapping Techniques:

1) **Associative Mapping** also called Fully Associative Mapping
2) **Direct Mapping** also called One-Way Set Associative Mapping
3) **Set Associative Mapping** also called Two-Way Set Associative Mapping

ASSOCIATIVE MAPPING (FULLY ASSOCIATIVE MAPPING)

During memory operations, Blocks are loaded from Main Memory to Cache Memory.
Cache Mapping decides which block of Main Memory comes into which block of Cache Memory.
Fully Associative Mapping technique states:
Any block of Main Memory can be mapped at Any available block of Cache Memory.
There are no rules restricting the mapping at all.
This means the Full Cache is available for mapping hence the name Fully Associative.

Consider Pentium Processor Cache

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in Main Mem:	Size of Main Memory (2^{32}) ÷ Size of Block (2^5) = 2^{27}
No. of Blocks in Cache Mem:	Size of Cache Memory (2^{13}) ÷ Size of Block (2^5) = 2^8 = **256**
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
A block of Cache Memory can **contain any block** of main memory out of a possible 2^{27} **blocks**.
Hence, the **Tag** next to every block in Cache Memory must be of **27 bits**.

Searches:
A block of main Memory can be **mapped into any block** of Cache Memory out of **256 Blocks**.
Hence, we need to do **256 searches** in Cache Memory.

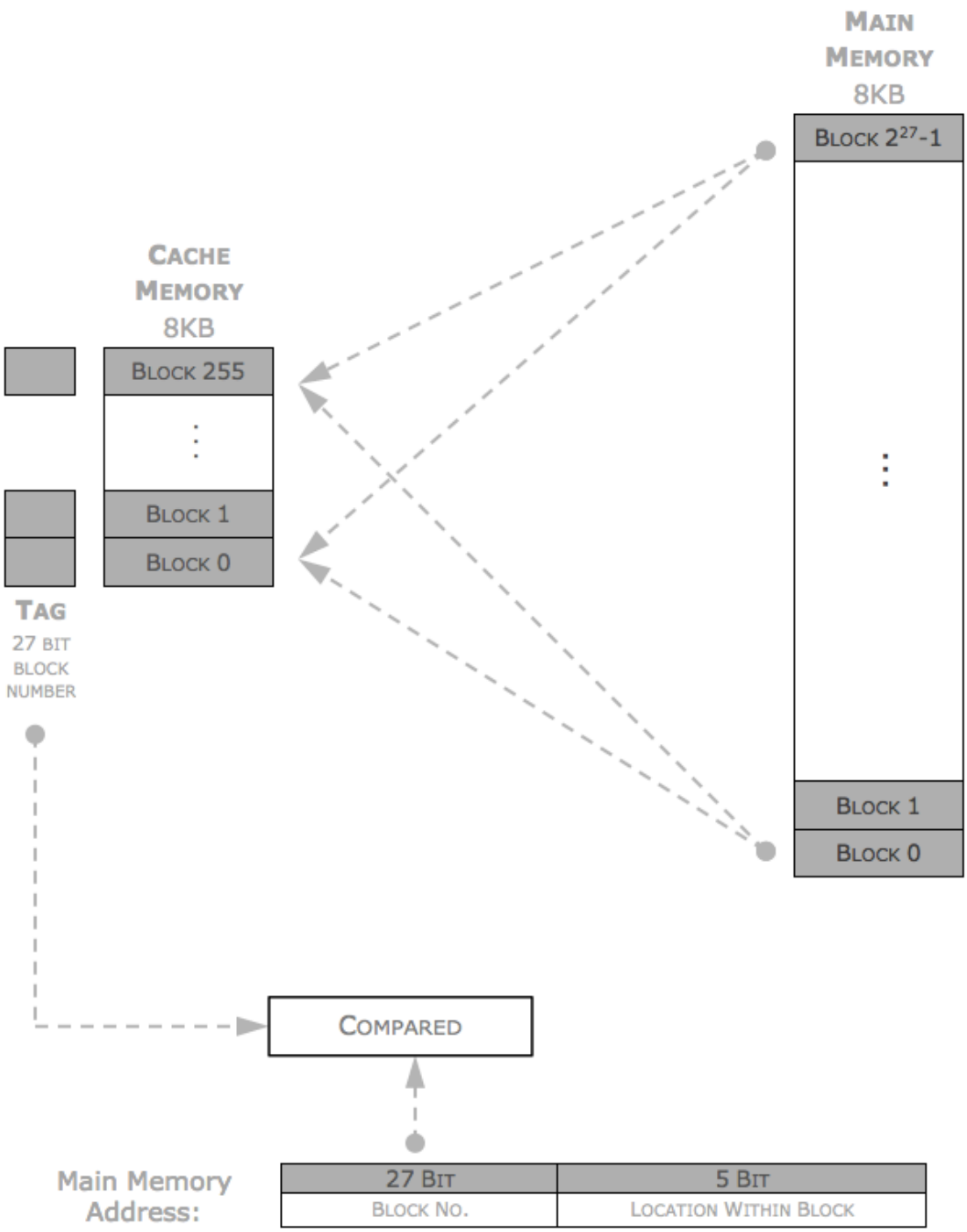
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	27 BIT	5 BIT
	Block No.	Location Within Block

This 27 bit Block number is the block number we need to search.
The Tag of each cache block also contains a 27-bit block number.
This is the block number that's present in that respective cache block.
These two block numbers are compared. **If they are equal, it's a HIT.**
If not equal, the search is repeated with the Tag of the next cache block.
This is done a total of 256 times as there are 256 blocks in Cache Memory.
If none of them match with the block number we are searching, then it's a Miss.

Advantage
Since the full cache is available for mapping, it causes maximum utilization of Cache Memory hence gives the **Best Hit Ratio.**

Drawback
Tag Size too big: **27bits.**
Searches are too many: **256.**


MAIN MEMORY
8KB
BLOCK 2^{27}-1
BLOCK 1
BLOCK 0
CACHE MEMORY
8KB
BLOCK 255
BLOCK 1
BLOCK 0
TAG
27 BIT BLOCK NUMBER
COMPARED
Main Memory Address:
27 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

DIRECT MAPPING (ONE WAY SET ASSOCIATIVE MAPPING)

Direct Mapping technique states:
Any block of Main Memory can only be mapped at ONE block of Cache Memory.
Since there is only one way of Mapping, its also called One Way Set Associative Mapping.
We treat the entire Cache as One Set.
The Main Memory is divided into Sets which are then subdivided into Blocks.
A Block of Main Mem. (of any set), can only be mapped into the same Block No. in Cache Mem.
This means, Block 0 of Main Memory (of any set), can only be mapped into Block 0 of Cache Memory.
In other words, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set.

Consider Pentium Processor Cache

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13} … this is treated as One Set
Hence Size of Set:	**8KB** = 2^{13}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in a set:	Size of Set (2^{13}) ÷ Size of Block (2^5) = 2^8 = **256**
No. of Sets in Main Mem:	Size of Main Mem (2^{32}) ÷ Size of Set (2^{13}) = 2^{19}
No. of Sets in Cache Mem:	1
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
Since, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set, the Tag has to only indicate the Set No. of Main Memory, from which the block is present.
As Main Memory has 2^{19} sets, the **Tag** size is **19 bits**.

Searches:
If we need Block 0 if Main Memory, we only need to search Block 0 of Cache Memory.
Hence, we need to do only **1 search** in Cache Memory to know if it is a Hit or a Miss.

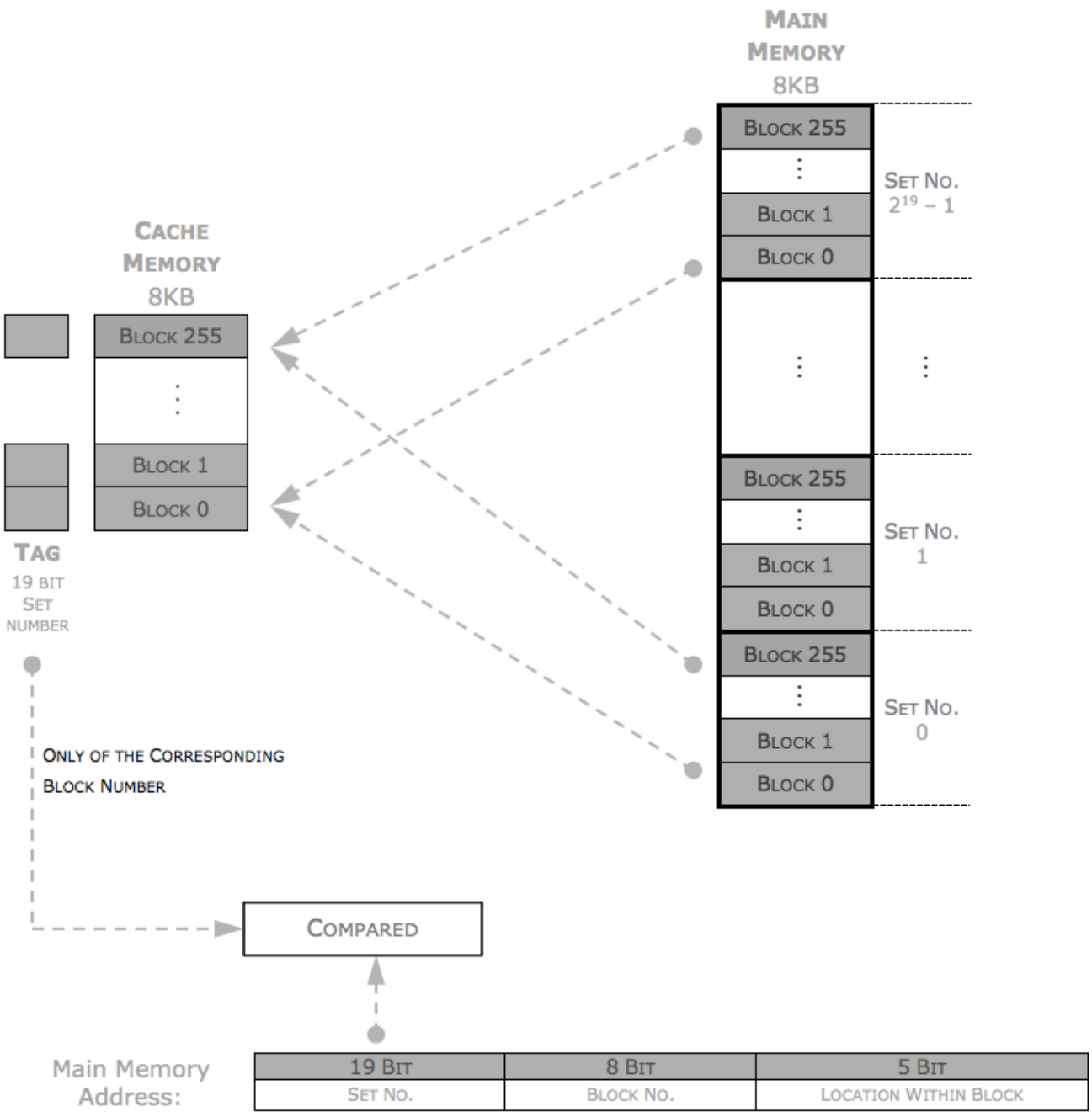
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	19 BIT	8 BIT	5 BIT
	Set No.	Block No.	Location Within Block

First we look at the block no. we need, to know where we need to search.
We then look at the Set No. that we need and compare it with the Tag of the corresponding Block no in the Cache Memory.
Assume the Main Memory address is 5:0:6. This means we need location 6 of Block 0 of set 5.
We go to Block 0 of Cache Memory.
It has a Tag, which gives the Set No of Main Memory whose Block 0 is present in Cache Memory.
These two set numbers are compared. **If they are equal, it's a HIT, else it's a Miss.**

Advantage
In **1 Search** we know if it is a Hit or a Miss. **Tag Size** = **19 bits**.

Drawback
Since the method is **very rigid**, the **Hit Ratio drops tremendously.**


MAIN MEMORY
8KB
BLOCK 255
BLOCK 1
BLOCK 0
SET NO. $2^{19} - 1$
BLOCK 255
BLOCK 1
BLOCK 0
SET NO. 1
BLOCK 255
BLOCK 1
BLOCK 0
SET NO. 0
CACHE MEMORY
8KB
BLOCK 255
BLOCK 1
BLOCK 0
TAG
19 BIT SET NUMBER
ONLY OF THE CORRESPONDING BLOCK NUMBER
COMPARED
Main Memory Address:
19 BIT
SET NO.
8 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

SET ASSOCIATIVE MAPPING (TWO WAY SET ASSOCIATIVE MAPPING)

Two way Set Associative Mapping technique states:
A block of Main Memory can only be mapped into the same corresponding Block No. of Cache Memory, in any of the two sets.
Since there are two ways of Mapping, its called Two Way Set Associative Mapping.
We treat the entire Cache as **Two Sets**. The Main Memory is divided into Sets, subdivided into Blocks.
A Block of Main Mem. (of any set), can only be mapped into the same Block No. in Cache Memory again of any set. This means, Block 0 of Main Memory (of any set), can only be mapped into Block 0 of Cache Memory, into one of its two sets.
In other words, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set.

Consider Pentium Processor Cache (This is Actually how Pentium's Cache is implemented)

Size of Main Mem:	**4GB** = 2^{32}
Size of Cache Mem:	**8KB** = 2^{13} … this is treated as Two Sets
Hence Size of Set:	**4KB** = 2^{12}
Size of Cache Block (Line):	**32** bytes (words) = 2^5
No. of Blocks in a set:	Size of Set (2^{12}) ÷ Size of Block (2^5) = 2^7 = **128**
No. of Sets in Main Mem:	Size of Main Mem (2^{32}) ÷ Size of Set (2^{12}) = 2^{20}
No. of Sets in Cache Mem:	2
Main Mem address:	**32 bits** (because main Mem is of **4GB** = 2^{32})

Tag Size:
Since, Block 0 of Cache Memory can only contain Block 0 of Main Memory but of any Set, the Tag has to only indicate the Set No. of Main Memory, from which the block is present.
As Main Memory has 2^{20} sets, the **Tag** size is **20 bits**.

Searches:
If we need Block 0 if Main Memory, we only need to search Block 0 of Cache Memory, but in any of the two sets. Hence, we need **2 searches** in Cache Memory to know if it is a Hit or a Miss.

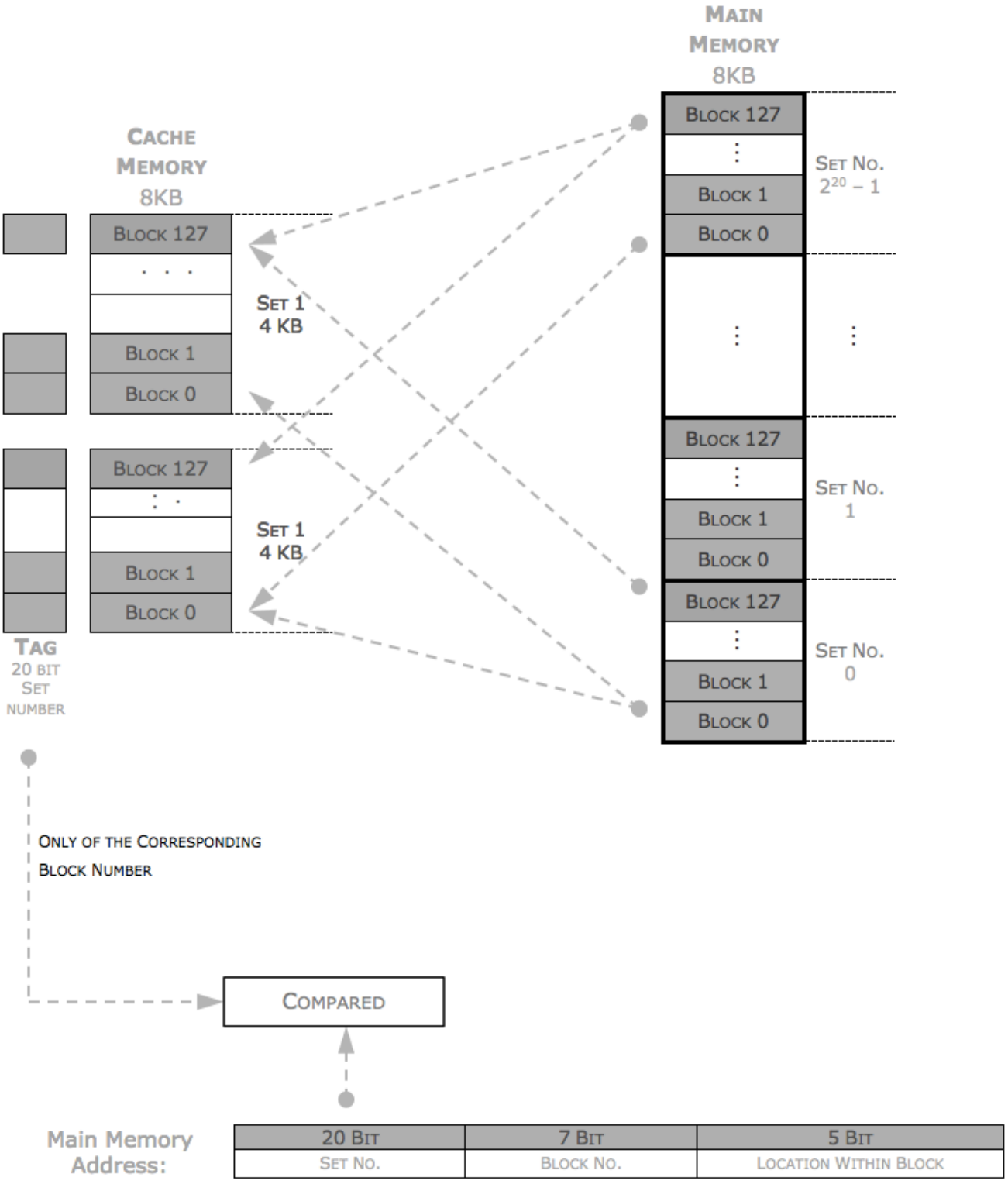
Method of Searching:
The Processor issues a 32 bit Main Memory address. It can be divided as:

Main Memory Address:	**20 BIT**	**7 BIT**	**5 BIT**
	Set No.	Block No.	Location Within Block

First we look at the block no. we need, to know where we need to search.
We then look at the Set No. that we need and compare it with the Tags of the corresponding Block no in the Cache Memory, in any of the two sets.
Assume the Main Memory address is 5:0:6. This means we need location 6 of Block 0 of set 5.
We go to Block 0 of Cache Memory, in both sets.
It has a Tag, which gives the Set No of Main Memory whose Block 0 is present in Cache Memory.
These required Set no. is compared with the two Tags. **If found in any of the 2, it's a HIT, else Miss.**

Advantage
In **2 Searches** we know if it is a Hit or a Miss. **Tag Size** = **20 bits**.

Drawback
Since the method is **flexible**, it significantly **increases** the **Hit Ratio.**


MAIN MEMORY
8KB
CACHE MEMORY
8KB
BLOCK 127
BLOCK 1
BLOCK 0
SET 1
4 KB
SET 1
4 KB
SET NO.
$2^{20} - 1$
SET NO.
1
SET NO.
0
TAG
20 BIT
SET
NUMBER
ONLY OF THE CORRESPONDING
BLOCK NUMBER
COMPARED
Main Memory
Address:
20 BIT
SET NO.
7 BIT
BLOCK NO.
5 BIT
LOCATION WITHIN BLOCK

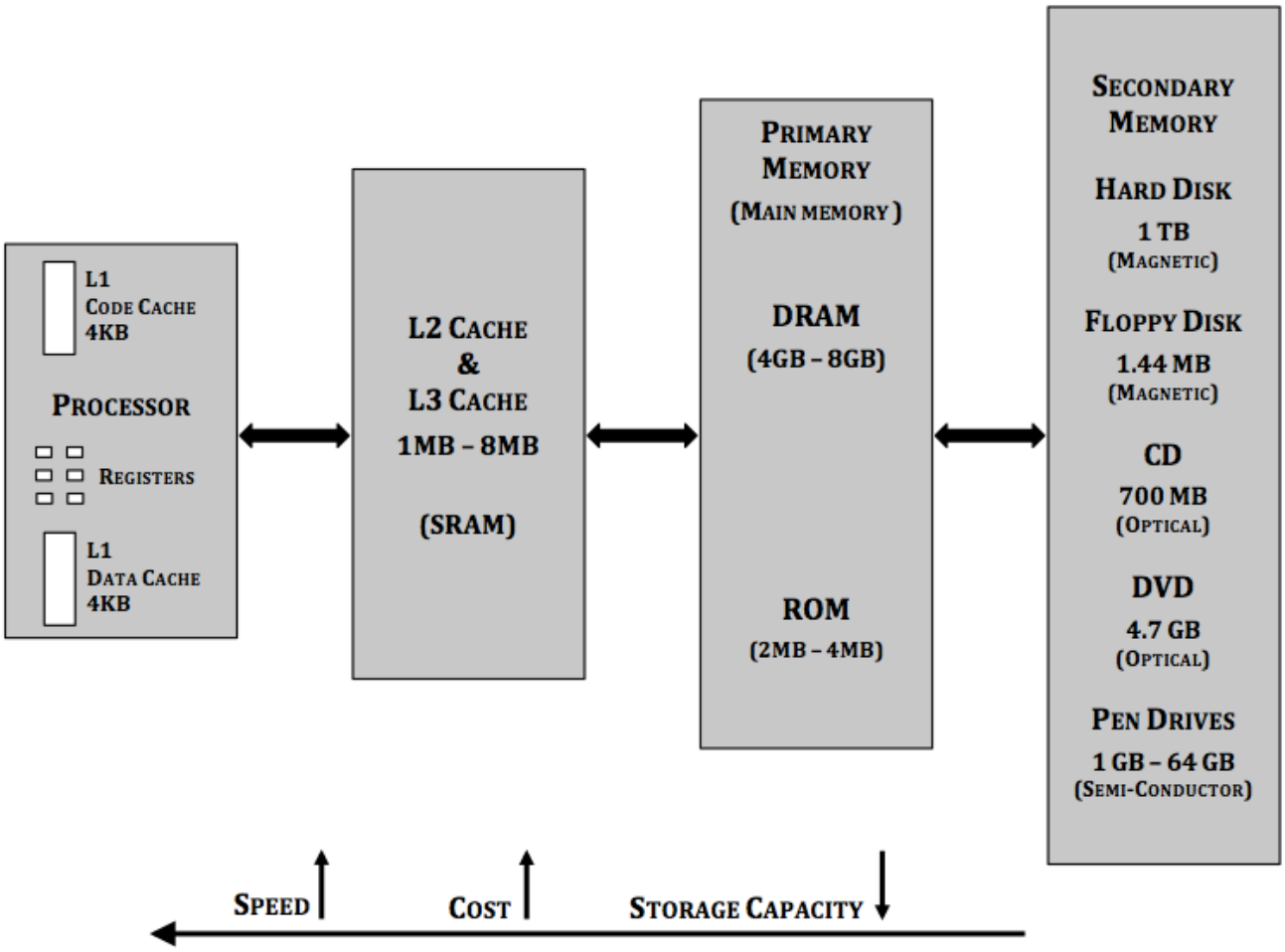
Expanding the logic of set associative cache further, we can derive the following conclusion:

NO OF WAYS	NO OF SEARCHES	TAG SIZE	NO OF BLOCKS IN A SET
2 Way	2	20 bits	128
4 Way	4	21 bits	64
8 Way	8	22 bits	32
16 Way	16	23 bits	16
32 Way	32	24 bits	8
64 Way	64	25 bits	4
128 Way	128	26 bits	2
256 Way	256	27 bits	1
This becomes exactly the same as Fully Associative: 256 Searches, 27-bit Tag			

MEMORY

MEMORY HIERARCHY

1) The purpose of any memory device is to **store programs and data**.
2) **Several types** of memory devices are used in the computer forming a **Memory Hierarchy**.
3) Each plays a specific role contributing to the **speed, cost effectiveness, portability** etc.



REGISTERS

1) Registers are **present inside the processor**.
2) They are basically a **set of flip-flops**.
3) They store **data and addresses** and can **directly take part** in **arithmetic and logic operations**.
4) They are very small in size typically **just a few bytes**.

PRIMARY MEMORY

1) It is the original form of memory also called as **Main memory**.
2) It comprises of **RAM and ROM**, both are **Semi-Conductor** memories. (chip memories)
3) **ROM is non-volatile**.
   It is used is storing permanent information like the **BIOS program**.
   It is typically of **2 MB - 4 MB** in size.
4) **RAM is writable** and hence is used for **day-to-day operations**.
   Every file that we access from secondary memory, is **first loaded into RAM**.
   To provide large amount of working space RAM is **typically 4 GB - 8 GB**.

SECONDARY MEMORY

1) The main purpose of Secondary Memory is to **increase the storage capacity, at low cost**.
2) Its biggest component is the **Hard Disk**.
   This is where all the files inside a computer **are stored**.
3) It is **writeable as well as non-volatile**.
4) Typical size of a **HD is 1 TB**.
5) Disk memories are much **slower than chip memories** but are also **much cheaper**.

PORTABLE SECONDARY MEMORY

1) These are required to **physically transfer files** between computers.
2) **Floppy Disk**: It is a **magnetic form** of storage. Typical **Size is 1.44 MB**.
3) **CD**: It is an **optical form** of storage. Typical **Size is 700 MB**.
4) **DVD**: It is an **optical form** of storage. Typical **Size is 4.7 MB**.
5) **Pen Drives & Memory Cards**: It is a **semi-conductor form** of storage.
   It is composed of FLASH ROM.
   It's a special type of ROM that's writable as well as non-volatile.
   Typical **Size ranges form 1 GB - 64 GB** depending upon the cost.

CACHE MEMORIES

1) It is the **fastest form of memory** as it uses SRAM (Static RAM).
2) The Main Memory uses DRAM (Dynamic RAM).
3) **SRAM uses flip-flops and hence is much faster than DRAM which uses capacitors.**
4) But SRAM is also **very expensive** as compared to DRAM.
5) Hence **only the current portion of the file** we need to access is copied from Main Memory (DRAM) to Cache memory (SRAM), to be directly accessed by the processor.
6) This gives **maximum performance and yet keeps the cost low**.
7) Typical size of Cache is around **2 MB – 8MB**.
8) If **code and data** are in the **same cache** then it is **unified cache** else its **called split cache**.
9) Depending upon the location of cache, it is of three types: L1, L2 and L3.
10) **L1** cache is **present inside the processor** and is a **split cache** typically **4-8 KB**.
11) L2 is present on the **same die as the processor** and is a **unified cache** typically **1 MB**.
12) L3 is present **outside the processor**. It is also **unified** and is typically of **2-8 MB**.

MEMORY CHARACTERISTICS

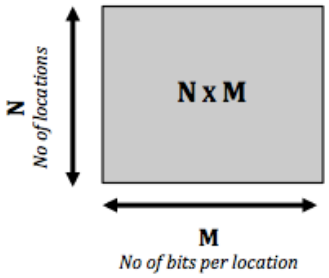
1) Location

Based on its physical location, memory is classified into three types.

- **On-Chip:** This memory is present **inside the CPU**. E.g.:: Internal Registers and **L1 Cache**.
- **Internal:** This memory is present **on the motherboard**. E.g.:: **RAM**.
- **External:** This memory is **connected to the motherboard**. E.g.:: **Hard disk**.

2) Storage Capacity

This indicates the **amount of data stored** in the memory.
Obviously it should be **as large as possible**.
It is represented as N x M.
Here,
N = **Number of memory locations** (no of words)
M = **Number of pits per memory location** (word size)
E.g.:: (4K x 8) means there are 4K locations of 8-bits each.



3) Transfer Modes

Data can be accessed from memory in two different ways.

- **Word Transfer:** Here, if CPU needs some data, it will transfer only that amount of data.
  E.g.:: Data accessed from **L1 Cache.**
- **Block Transfer:** Here, if CPU needs some data, it will transfer an entire block containing that data. This makes further access to remaining data of this block much faster. This is based on Principle of Spatial Locality. A processor is most likely to access data near the current location being accessed.
  E.g.:: On a **cache miss**, processor goes to **main memory** and copies a **block** containing that data.

4) Access Modes

Memories can allow data to be accessed in two different ways.

- **Serial Access:** Here locations are accessed one by one in a **sequential manner**.
  The access time depends on how far the target location is, from the current location.
  Farther the location, **more** will be its **access time**.
  E.g.:: Magnetic tapes.
- **Random Access:** Here **all locations** can be directly accessed in any **random order**.
  This means **all locations** have the **same access time** irrespective of their address.
  E.g.:: Most modern memories like RAM.

5) Physical Properties

There are various Physical attributes to memory.

- **Writeable: Contents** of the memory **can be altered**. E.g.:: **RAM**
- **Non-Writeable: Contents** of the memory **cannot be altered**. E.g.:: **ROM**
- **Volatile: Contents** of the memory are **lost** when power is **switched off**. E.g.:: **RAM**
- **Non-Volatile: Contents** of the memory are **retained** when power is **switched off**. E.g.:: **ROM**

Most secondary memories like Hard disk are Writable as well as non-volatile.

6) Access Time (t_A)

It is the time taken between **placing the request** and **completing the data transfer**.
It should be as **less as possible**.
It is also known as **latency**.

7) Reliability

It is the **time** for which the memory is expected to **hold the data without any errors**.
It is measured as **MTTF: Mean Time To Failure**.
It should be as **high as possible**.

8) Cost

This indicates the **cost of storing data** in the memory.
It is expressed as **Cost/bit**.
It must be **as low as possible**.

9) Average Cost

It is the total cost per bit, for the entire memory storage.
Consider a system having **two memories M_1 (RAM) & M_2 (ROM)**
If **C_1** is the cost of memory M_1 of size **S_1**
& **C_2** is the cost of memory M_2 of size **S_2**

Then the average cost of the memory is be calculated as:
$C_{AVG} = (C_1 S_1 + C_2 S_2)/ (S_1 + S_2)$

Small sizes of **expensive** memory and **large** size of **cheaper** memory **lowers** the **average cost**.

10) Hit Ratio (H)

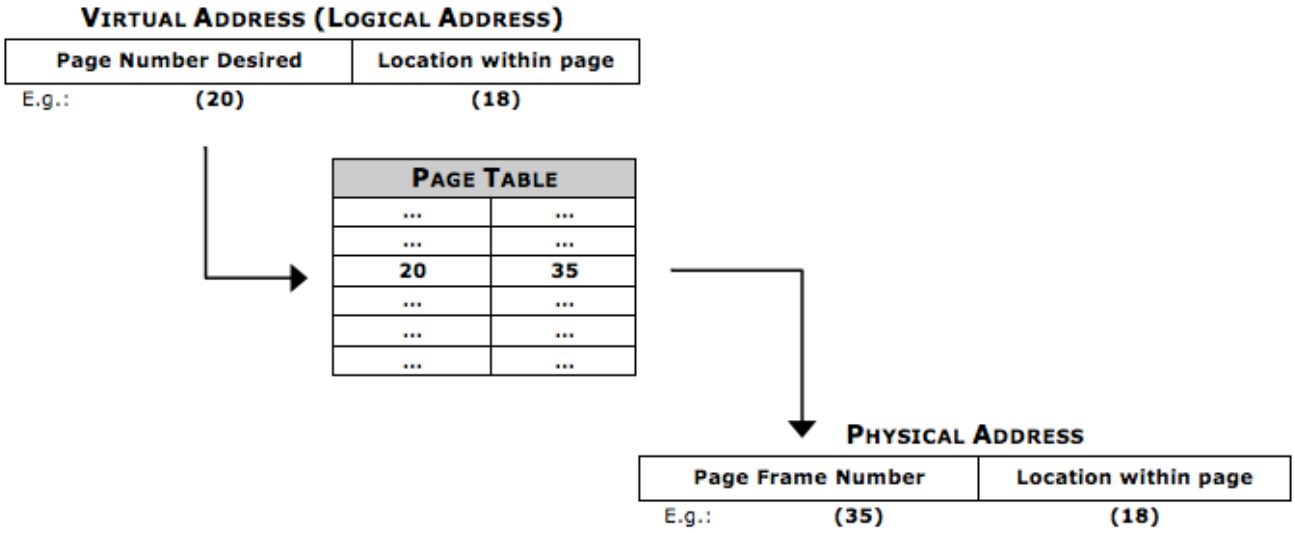
Consider two memories M_1 and M_2.
M_1 is **closer** to the processor E.g.:: **RAM**, than **M_2** E.g.:: **Hard disk**.
If the **desired data is found in M_1**, then it is called a **Hit**, else it is a **Miss**.
Let **N_1** be the number of **Hits** and **N_2** the number of **Misses**.
The **Hit Ratio** H is defined as **number of hits divided by total attempts**.
$H = (N_1) / (N_1 + N_2)$
It is expressed s a percentage.
H can never be 100%. In most computers it is maintained around 98%.

From the above discussion it is clear that no single memory can satisfy all the characteristics, **hence we need a hierarchy of memories**.
Cache memories are the **fastest** but also the **most costly**.
Hard disk is **writeable** as well as **non volatile** and is also very **inexpensive**, but is much **slower**.
CD/DVD etc. are needed for **portability**.
ROM is **nonvolatile**, and is used for **storing BIOS**.
DRAM is **writable**, f**aster than hard disk** and **cheaper than SRAM** hence forms **most part of Main Memory**.

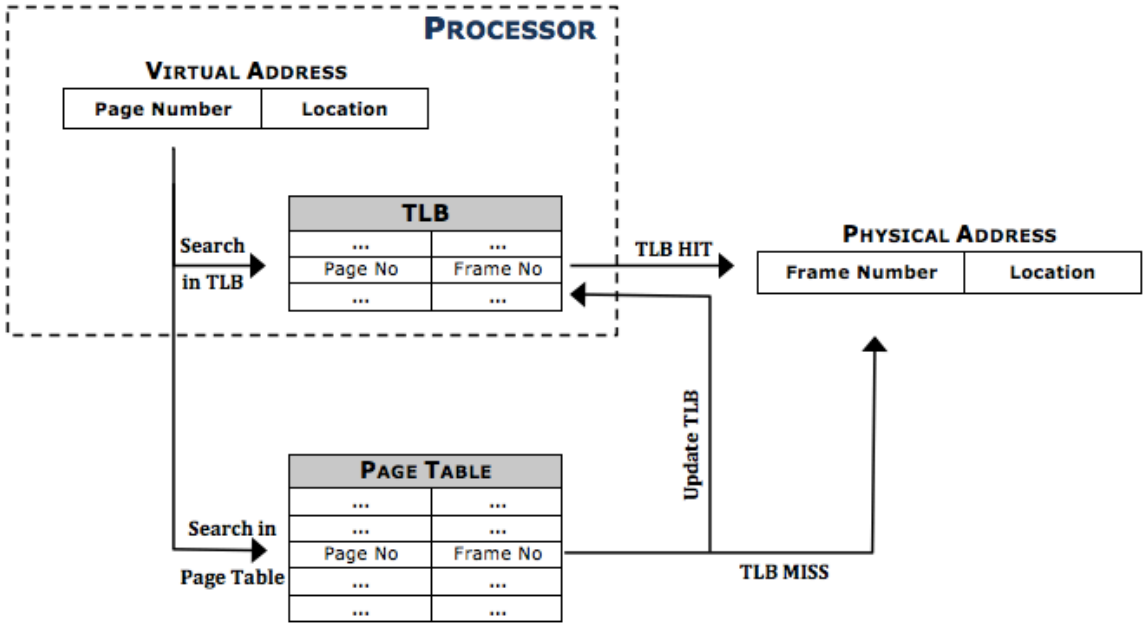
VIRTUAL MEMORY MANAGEMENT USING PAGING

1) The **Virtual Memory space** is divided into **equal size blocks** called **"Pages"**.
2) The **Physical Memory space** (also called Main Memory) is also divided into **equal size blocks** called **Page Frames** (also simply called pages).
3) **Pages are loaded** from Virtual Memory **into any** available page frame of Physical Memory.
4) Consider example of 80386 Processor. Physical Memory = 4 GB. Page Size = 4 KB.
   No of pages in Physical Memory = 4 GB / 4 KB = 1M. ($2^{32} / 2^{12} = 2^{20}$)
5) When a page is needed, Processor **checks** if the **desired page** is **present** in the **Physical Memory**.
6) If found, it is called a **"HIT"** and the operation is performed on the Physical Memory.
7) If the **desired page** is **not present** in the **Physical Memory** its called a **MISS** or a **Page Fault.**
8) On a Page Fault the desired page is loaded form Virtual Memory into Physical Memory.
9) If no empty page frame is available in Physical Memory, then a **"Page Replacement"** is performed.
10) An old page in Physical Memory is replaced with a new desired page from the Virtual Memory.
11) Popular **algorithms** like **FIFO** (First in first out), **LRU** (Least recently used) or **LFU** (Least frequently used) are used to determine which page of the Physical Memory must be replaced.
12) If there are **too many page faults**, the system is said to be **Thrashing**. This must be avoided.
13) While replacing, the **"Dirty Bit"** is checked to determine if a page is **modified** in Physical Memory.
14) **If Dirty bit = 1, then the page has been modified** (is "Dirty") and hence must be copied back into Virtual Memory before being replaced else the modified information will be lost.
15) **If Dirty bit = 0, then the page can be directly replaced**.
16) Since **any page** of Virtual Memory can be **loaded into any page frame** of Physical Memory, a **"Page Table"** is required to give the **mapping between** Virtual Memory **page number** and Physical Memory **page frame number**.
17) **Virtual Address** can be divided in two parts: **Desired Page Number | Location in page**
18) **Physical Address** can be divided in two parts: **Present Page Frame Number | Location in page**
19) The Page Table relates the desired page number to the page frame number (if present) in the Physical Memory. This is how address translation is performed.
20) **But this process is very slow**. To access any location, we must first access the corresponding entry in the page table, then access the page.
21) **Page Table is also in the memory**. This means, to access any memory location we must make an **additional trip** to the memory just to access the page table.
22) To speed up the process a **"Translation Look-aside Buffer"** is used (called TLB).
23) **The TLB is an on chip cache, present inside the processor.**
24) **It stores 32 most recently used Page table entries.**
25) This makes subsequent access to these pages (whose information is cached in the TLB) **much faster as there is no need to access the page table**. Processor can directly obtain the starting address of the page frame from the TLB and hence directly access the page.
26) Due to principle of **"Locality of reference"** most systems get a **Hit ratio of >98%** on the TLB, thus making the operations very fast.
27) An application may require several pages. All are not loaded into Physical Memory at once. Instead, pages are loaded as and when they are required by the processor. This is called **Demand Paging**.

PAGING MECHANISM:



TRANSLATION LOOK-ASIDE BUFFER (TLB)



PAGING V/S SEGMENTATION

	PAGING	SEGMENTATION
1	Paging is a **physical division** of the memory.	Segmentation is a **logical division** of the memory.
2	Pages are of a **fixed size**.	Segments are of **variable sizes**.
3	**Fragmentation** (wastage of unused space) **is high** as pages are of **fixed size.**	**Fragmentation is low** as segments are of **variable size.**
4	Completely **hidden from programmer**.	**Controlled by programmer.**
5	Does **not** offer **much control over protection mechanism**	As segments are controlled by the programmer, we can assign **different privilege levels to segments**. Hence provides better control over protection mechanism.
6	**Does not differentiate between code and data.**	Code and data segments are **treated differently**.
7	Pages are loaded based on **algorithms** such as **FIFO, LRU, LFU** etc.	Segments are loaded based on **algorithms** such as **First Fit, Best Fit** etc.
8	Does **not allow sharing** of code or data between different processes/ tasks.	Segments can be either global or local. **Global segments can be shared** among all tasks.

PAGE REPLACEMENT NUMERICALS *(Very Important)*

1) Consider Main Memory has 3 page frames (0,1,2).
Processor requires pages from Virtual Memory in the following sequence of page numbers:
2,3,2,1,5,2,4,5,3,2,5,2. Show and compare the implementation of FIFO, LRU and LFU.

FIFO	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.25
Frame 0	2*	2*	2*	2*	5	5	5*	5*	3	3	3	3*	
Frame 1		3	3	3	3*	2	2	2	2*	2*	5	5	
Frame 2				1	1	1*	4	4	4	4	4*	2	
			HIT					**HIT**		**HIT**			

LRU	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.42
Frame 0	2	2	2	2	2	2	2	2	3	3	3	3	
Frame 1		3	3	3	5	5	5	5	5	5	5	5	
Frame 2				1	1	1	4	4	4	2	2	2	
			HIT			**HIT**		**HIT**			**HIT**	**HIT**	

LFU	2	3	2	1	5	2	4	5	3	2	5	2	Hit Ratio = 0.50
Frame 0	2 (1)	2 (1)	2 (2)	2 (2)	2 (2)	2 (3)	2 (3)	2 (3)	2 (3)	2 (4)	2 (4)	2 (5)	
Frame 1		3 (1)	3 (1)	3 (1)	5 (1)	5 (1)	5 (1)	5 (2)	5 (2)	5 (2)	5 (3)	5 (3)	
Frame 2				1 (1)	1 (1)	1 (1)	4 (1)	4 (1)	3 (1)	3 (1)	3 (1)	3 (1)	
			HIT			**HIT**		**HIT**		**HIT**	**HIT**	**HIT**	

2) Consider Main Memory has 4 page frames (0,1,2,3)
Processor requires pages from Virtual Memory in the following sequence of page numbers:
7,5,3,2,1,0,4,1,6,7,4,2. Show and compare the implementation of FIFO, LRU and LFU.

FIFO	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

LRU	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

LFU	7	5	3	2	1	0	4	1	6	7	4	2	Hit Ratio = 0.16
Frame 0													
Frame 1													
Frame 2													
Frame 3													

OPTIMAL REPLACEMENT ALGORITHM

1. Another proposed replacement algorithm is called "**Optimal Replacement Algorithm**" (**OPT**)
2. We need to know beforehand, the order in which, pages will be used in the near future.
3. **The pages that will be used sooner, will be retained.**
4. **The pages that will not be used for the longest time will be replaced.**
5. Of course it is impossible to predict the pages to be used in the future.
6. But if we have had some **sample runs of the program in a simulator** then using that data as a reference, we can make safe predictions of the behavior of the program.

Consider Main Memory has 3 page frames (0,1,2).
Calculate Hits and Misses and suggest the best algorithm out of **FIFO, LRU and OPT**
Processor requires pages from Virtual Memory in the following sequence of page numbers:
4,7,3,0,1,7,3,8,5,4,5,3,4,7. (Sem 4 Comps IT Dec 2015 Exam question – 10 marks)

FIFO	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4*	4*	4*	0	0	0*	3	3	3*	4	4	4	4	4*
Frame 1		7	7	7*	1	1	1*	8	8	8*	8*	3	3	3
Frame 2			3	3	3*	7	7	7*	5	5	5	5*	5*	7
	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	**HIT**	MISS	**HIT**	MISS

Total Attempts: 14. Hits = 2. Misses (Page Faults) = 12. **Hit Ratio = 2/14 = 0.117**

LRU	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4	4	4	0	0	0	3	3	3	4	4	4	4	4
Frame 1		7	7	7	1	1	1	8	8	8	8	3	3	3
Frame 2			3	3	3	7	7	7	5	5	5	5	5	7
	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	**HIT**	MISS	**HIT**	MISS

Total Attempts: 14. Hits = 2. Misses (Page Faults) = 12. **Hit Ratio = 2/14 = 0.117**

OPT	4	7	3	0	1	7	3	8	5	4	5	3	4	7
Frame 0	4	4	4	0	1	1	1	8	5	5	5	5	5	7
Frame 1		7	7	7	7	7	7	7	7	4	4	4	4	4
Frame 2			3	3	3	3	3	3	3	3	3	3	3	3
	MISS	MISS	MISS	MISS	MISS	**HIT**	**HIT**	MISS	Miss	Miss	**HIT**	**HIT**	**HIT**	Miss

Total Attempts: 14. Hits = 5. Misses (Page Faults) = 9. **Hit Ratio = 5/14 = 0.357**

BUS CONTENTION / BUS ARBITRATION / PRIORITY RESOLVING SCHEMES

In a Loosely coupled system all processors can use their local bus simultaneously. But the system bus can be used by only one module at a time. Hence there is contest for the system bus. This is called bus contention. It is resolved buy various arbitration schemes having different priority methods.

A) Daisy Chain Method

- **All** bus **masters use** the **same** line for Bus **Request**.
- **If** the **Bus Busy** line is **inactive**, the **Bus Controller gives** the **Bus Grant signal**.
- Bus **Grant** signal is **propagated serially** through all masters **starting** from **nearest** one.
- The bus **master**, which requires the system bus, **stops this signal**, **activates** the Bus **Busy** line and **takes control** of the system bus.

Advantage:

i. **Design** is **simple**.
ii. The **number of control lines is less**. Also **adding** new bus masters is **easy**.

Disadvantage:

i. **Priority** of bus masters is **rigid** and **depends** on the **physical proximity** of the bus masters with the bus arbiter i.e. The one nearest to the Bus Arbiter gets highest priority.
ii. Bus is granted **serially** and hence a **propagation delay** is induced in the circuit.
iii. **Failure of one** of the devices may **fail** the entire **system**.

B) Polling Method

- Here also **all bus masters** use the **same** line for **Bus Request**.
- Here the **controller generates** binary **address** for the master.
  Eg: To connect 8 bus masters we need 3 address lines ($\mathbf{2^3 = 8}$).
- In **response** to a **Bus Request**, the **controller "polls"** the bus masters by **sending** a **sequence** of bus master **addresses** on the address lines.Eg: 000,010,100,011 etc
- The selected **master activates** the **Bus Busy** line and **takes control** of the bus.

Advantage:

i. The **Priority** is **flexible** and can easily be **changed** by **altering** the **polling sequence**.
ii. If **one** module **fails**, the entire **system does not fail**.

Disadvantage:

i. **Adding** more bus masters is **difficult** as **increases** the number of **address lines** of the circuit. Eg: In the above circuit to add the 9^{th} Bus Master we need 4 address lines.

C) Independent Request Method

- Here, **all bus masters** have their **individual Bus Request** and **Bus Grant lines**.
- The **controller** thus **knows which master** has **requested**, so bus is granted to that master.
- **Priorities** of the masters are **predefined** so on **simultaneous** Bus **Requests**, the bus is **granted based** on the **priority**, provided the Bus Busy line is not active.
- The **Controller** consists of **encoder** and **decoder** logic for the priorities.

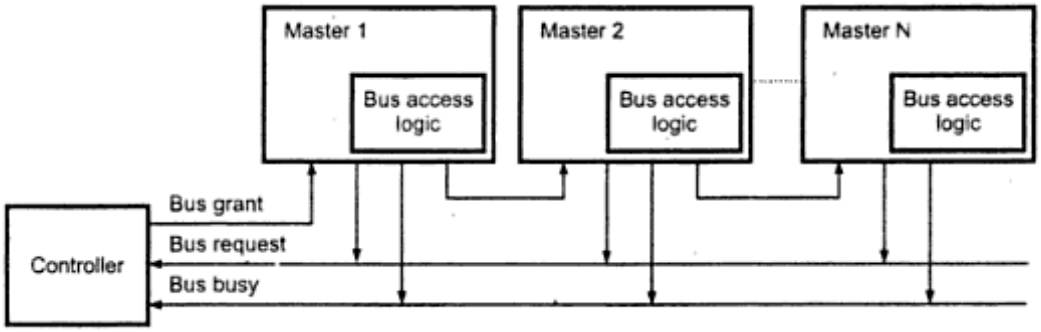
Advantage:

i. The Bus **Arbitration is fast**.
ii. The **speed of** Bus **Arbitration** is **independent** of the **number** of devices connected.

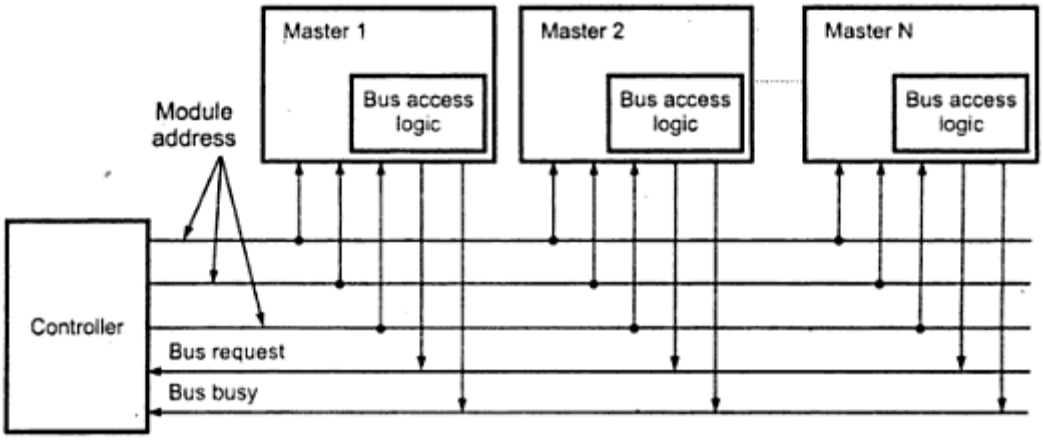
Disadvantage:

i. The **number of control lines required is more** (2n line required for n devices).

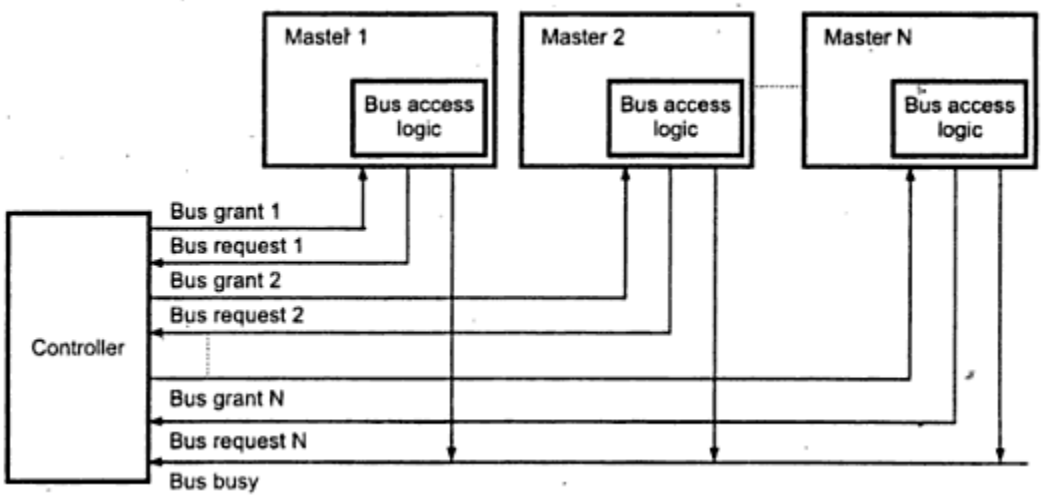
Daisy Chaining



Polling



Independent Requests

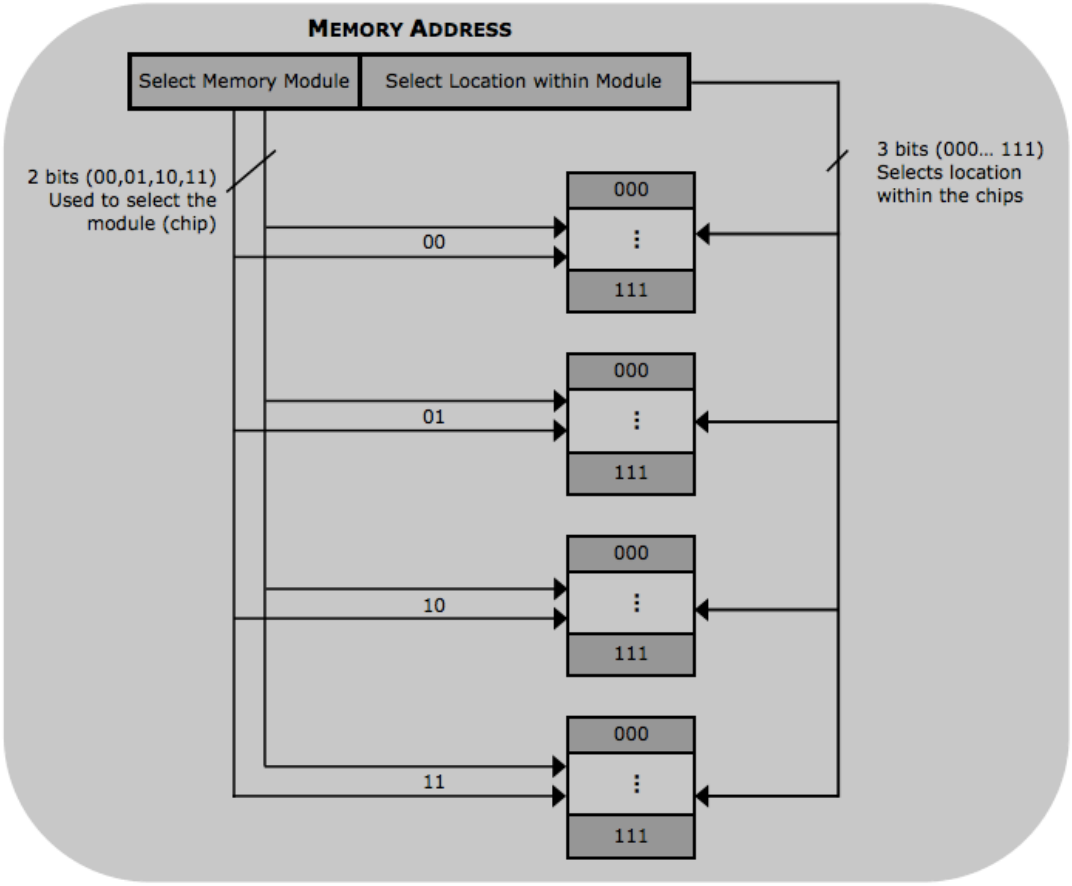


INTERLEAVED MEMORIES

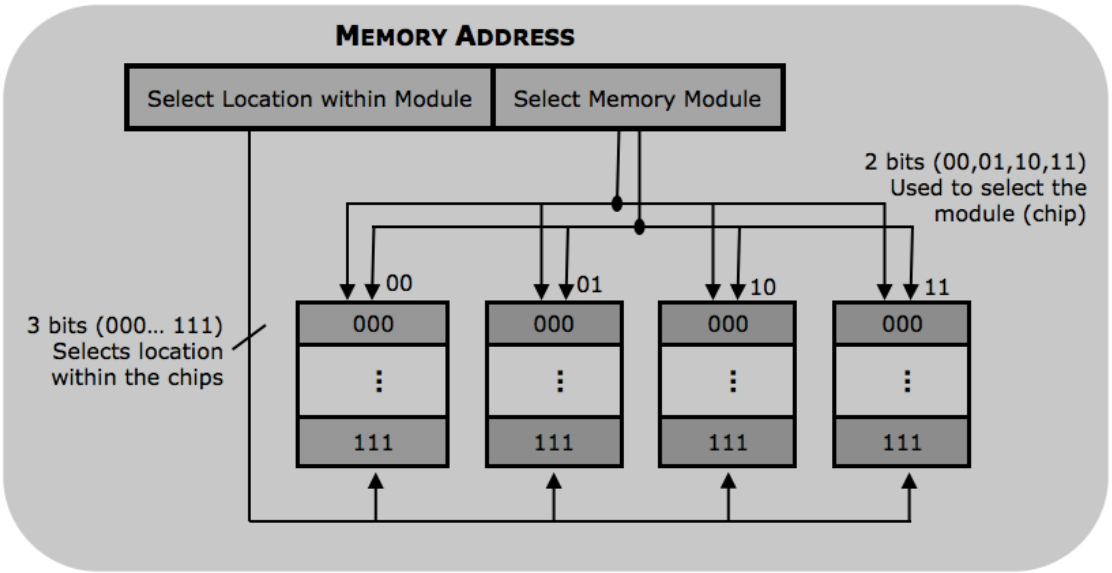
Memory Interleaving means **combining smaller memory modules to form one large memory.**
In most computer systems memory is not implemented as one single large module (chip).
Instead, it is formed by combining several smaller modules using the method of memory interleaving.
There are two interleaving techniques called Higher order and Lower order interleaving.

	HIGHER ORDER INTERLEAVING	LOWER ORDER INTERLEAVING
1	**Higher bits of the memory address are used to identify the memory module.**	**Lower bits of memory address are used to identify the memory module.**
2	Used to increase the **size of memory**.	Has **no effect** on size of memory.
3	Has **no effect** on the number of bits that can be transferred in one cycle.	**Increases the number of bits that can be transferred in one cycle**.
4	**Does not** increase the speed of the processor.	**Increases the speed** as it increases the number of bits transferred in 1 cycle.
5	**It is optional** and is only done when we need to increase the overall storage capacity.	**It is compulsory** in processors that need more bits in one cycle. E.g.:: 8086, 16-bit processor has two banks (Modules). 80386, 32-bit processor has 4 banks etc…
6	The **number of modules used are not fixed** and can be changed whenever required.	The **number of modules used is fixed** depending upon how many bits are needed in one cycle.
7	**Failure of one module does not effect the others.**	**Failure of one module effects all other modules** are data is "stripped" across modules.
8	**Reliability is high**	**Reliability is poor**
9	**Consecutive locations are in the same** module.	**Consecutive locations are in different** modules.
10	**Only one module** can be selected at a time.	**One or more modules can be selected simultaneously** depending upon the size of data needed to be accessed.

HIGHER ORDER INTERLEAVING



LOWER ORDER INTERLEAVING



3) DMA BASED I/O

DMA means **transferring data directly between memory and I/O.**
DMA transfers are **very fast** as compared to Processor based transfers due to two reasons.
1. They are **hardware based** so no time is wasted in fetching and decoding instructions.
2. Transfers are **directly between memory and I/O** without data going via the Processor.

To Perform a DMA transfer we need a **DMA Controller like 8237/ 8257.**
It is capable of taking control of the buses from the Processor.
The process is performed as follows.

1) By Default **Processor is the bus master**.
2) The DMA transfer parameters first initialized by the processor.
3) **Processor programs two registers inside the DMAC called CAR and CWCR** giving the starting address and the number of bytes to be transferred.
4) DMAC now ensures that the I/O device is ready for the transfer by checking the DREQ signal.
5) **If DREQ=1, then DMAC gives HOLD signal** to the Processor requesting control of the system bus.
6) **Processor releases control of the bus** after finishing the current machine (bus) cycle.
7) Processor **gives HLDA** informing DMAC that it is now the bus master.
8) **DMAC issues DACK#** (by default active low, but can be changed) to I/O device indicating that the transfer is about to begin.
9) Now DMAC **transfers one byte in one cycle**.
10) After every byte is transferred the **Address register and Count register are decremented by 1**.
11) This repeats till Count reaches **"0"** also called **Terminal Count**.
12) Now the **transfer is complete**.
13) DMAC **returns the system bus to Processor by making HOLD = 0**.
14) Processor once again **becomes bus master**.

Advantage of DMA
DMA transfers are very fast.

Drawback of DMA
DMAC becomes the bus master. Hence during DMA cycles, the processor cannot perform any operations as the bus is already being used for DMA. The processor remains in HOLD state.

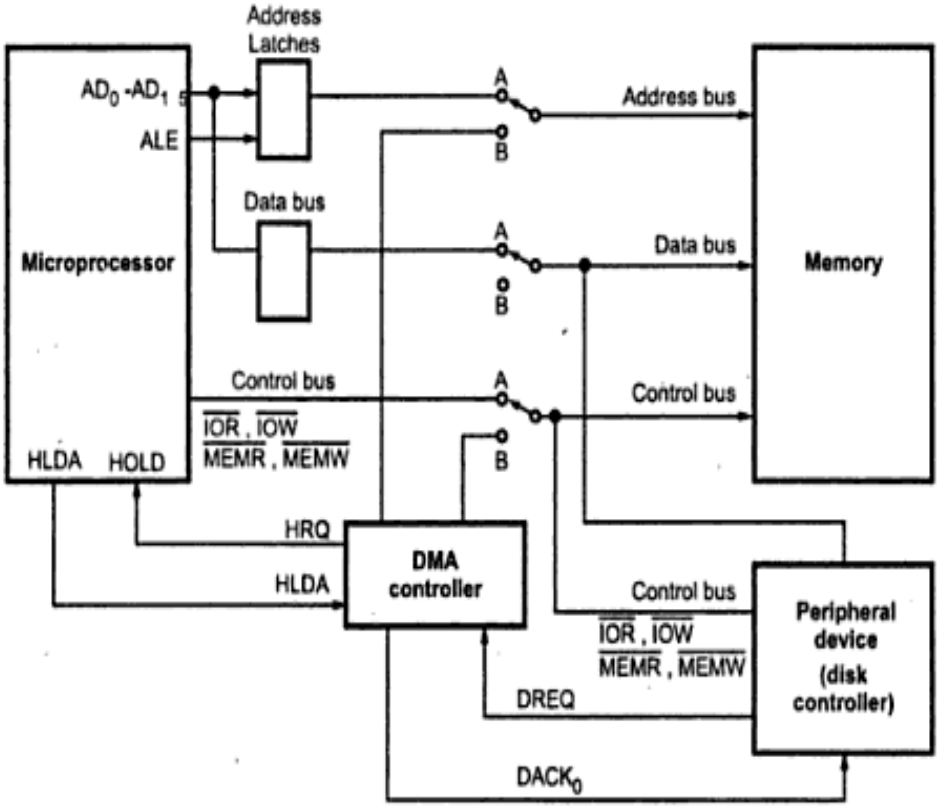
Difference between Interrupt Request and DMA request

When an interrupt occurs, the processor has to suspend the current program, execute the ISR and then **return to the next instruction** of the main program. Hence it is necessary that the **processor completes the current instruction** before servicing an interrupt request.

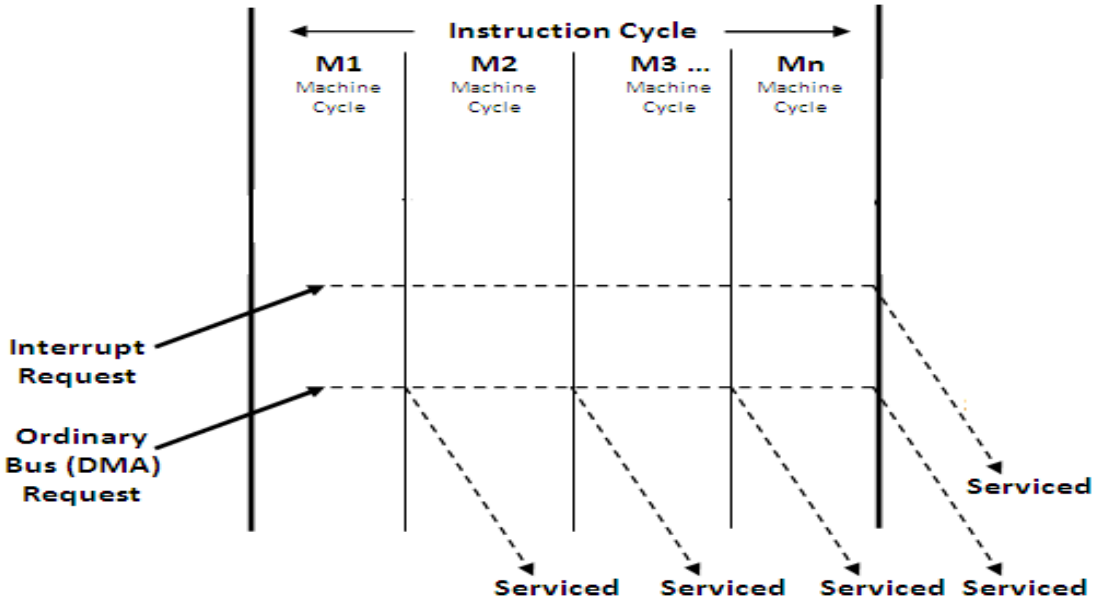
When a DMA request occurs, the processor has to simply relinquish (give away) control of the system bus and enter hold state. When ever it gets back the bus it can **resume from where it had left**. Hence the processor **need not finish the current instruction** before servicing a DMA request.
It simply has to **finish the current machine cycle**.

Hence Instruction cycles are Interrupt Breakpoints and Machine cycles are DMA breakpoints.

DMA Operation



Difference between Interrupt Request and DMA Request



TYPES / METHODS / TECHNIQUES OF DMA TRANSFERS

8237 has **four modes of data transfer**:

1) BLOCK TRANSFER MODE / BURST MODE.

In this mode, the DMAC is programmed to **transfer ALL THE BYTES** in one complete DMA operation.
After a byte is transferred, the **CAR and CWCR are adjusted** accordingly.
The **system bus** is **returned** to the **processor, ONLY after all the bytes are transferred.**
It is the **fastest** form of DMA but keeps the **processor inactive** for a long time.

2) SINGLE BYTE TRANSFER MODE/ CYCLE STEALING.

Once the DMAC becomes the bus master, it will transfer only **ONE BYTE** and return the bus to the processor. As soon as the processor performs one bus cycle, DMAC will once again take the bus back from the processor.
Hence both **DMAC and processor** are **constantly stealing bus cycles** from each other.
It is the **most popular** method of DMA, because it keeps the **processor active in the background**.
After a byte is transferred, the **CAR** and **CWCR** are **adjusted** accordingly.

3) DEMAND TRANSFER MODE.

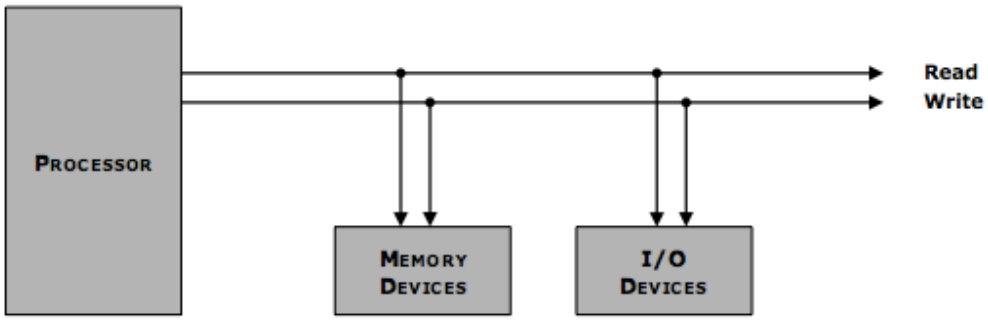
It is very **similar to Block Transfer**, **except** that the **DREQ must remain active throughout the DMA operation**.
If during the operation **DREQ goes low**, the **DMA operation is stopped** and the **busses** are **returned** to the **processor.** *#Please refer Bharat Sir's Lecture Notes for this ...*
In the meantime, the **processor** can **continue** with its own operations. **Once DREQ goes high again**, the **DMA operation continues** from where it had stopped.
This means, the transfer happens on demand from the I/O device, hence the name.

4) HIDDEN MODE / TRANSPARENT MODE.

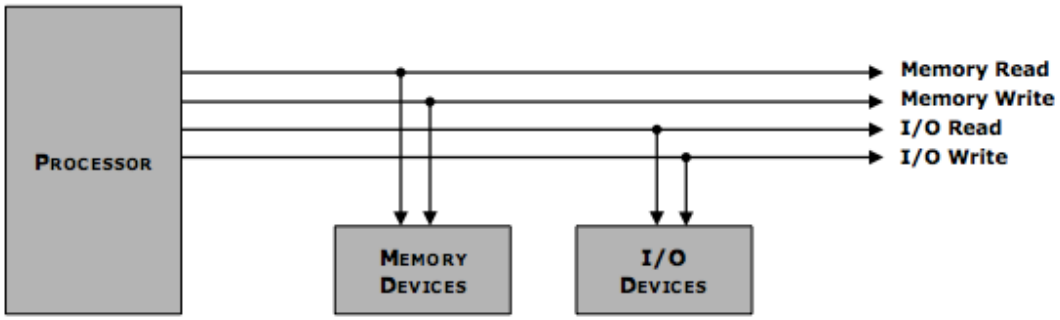
In this mode, **once the processor programs all parameters inside the DMAC, the DMAC** does not request the processor for the control of the bus. Instead it observes the processor. **It waits for the processor to enter idle state**. Once the processor is idle, the DMAC will take control of the bus and perform the Transfer. So the Transfer is **totally transparent to the processor** or hidden from the processor. Hence the name.

	MEMORY MAPPED I/O	I/O MAPPED I/O
1	**I/O devices are mapped into memory space.**	**I/O devices are mapped into I/O space.**
2	I/O devices are allotted **memory addresses**.	I/O devices are allotted **I/O addresses**.
3	Processor **does not differentiate** between memory and I/O. Treats I/O devices also like memory devices.	Processor **differentiates between I/O devices and memory**. It isolates I/O devices.
4	I/O addresses are as big as memory addresses. E.g.:: in 8085, **I/O addresses will be 16 bit** as memory addresses are also 16-bit.	I/O addresses are smaller than memory addresses. E.g.:: in 8085, **I/O addresses will be 8 bit** though memory addresses are 16-bit.
5	This allows us to increase the number of I/O devices. E.g.:: in 8085, we can access up to **2^{16} = 65536 I/O devices**.	This allows us to access limited number of I/O devices. E.g.:: in 8085, we can access only up to **2^{8} = 256 I/O devices**.
6	We can transfer data from I/O devices using **any instruction like MOV** etc.	We can transfer data from I/O device using **dedicated I/O instructions like IN and OUT ONLY**.
7	Data can be transferred using **any register** of the processor.	Data can be transferred only using a fixed register. E.g.:: in 8085 only **"A" register**.
8	We need **only two control** signals in the system: Read and Write.	We need **four control signals**: Memory Read, Memory Write and I/O Read and I/O Write
9	Memory addresses are big so **address decoding will be slower**.	I/O addresses are smaller so **address decoding will be faster**.
10	Address decoding will be **more complex and costly**.	Address decoding will be **simpler and cheaper**.

MEMORY MAPPED I/O



I/O MAPPED I/O

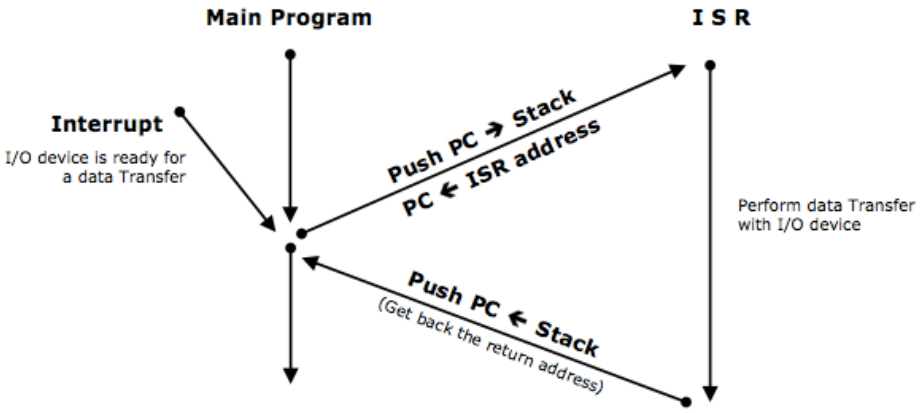


2) INTERRUPT DRIVEN I/O

1) In interrupt driven I/O, the transfer is not initiated by the processor.
2) Instead, **an I/O device which wants to perform a data transfer with the processor, must give an interrupt to the processor.**
3) An interrupt is a condition that makes the processor execute an ISR (Interrupt Service Routine).
4) In the ISR, processor will perform data Transfer with the I/O device.
5) This relieves the processor from periodically checking the status of every I/O device thereby saves as lot of time of the processor.
6) **The processor is free to carry on its own operations.**
7) Whenever a device wants to transfer data, it will interrupt the processor.
8) This is how many I/O devices Transfer data with the processor.
9) E.g.:: **Keyboard.** Instead of the processor checking all the time, whether a key is pressed, the keyboard interrupts the processor as an when we press a key. In the ISR of the keyboard, which is a part of the keyboard driver software, the processor will read the data from the keyboard.
10) **Hence interrupt driven I/O is much better than Polled I/O (Programmed I/O).**

INTERRUPT HANDLING MECHANISM

1) When an interrupt occurs, processor, firstly, finishes the current instruction.
2) It then suspends the current program and executes an ISR.
3) To do so, it Pushes the value of PC (address of next instruction), into the stack.
4) Now it loads the ISR address into PC and proceeds to execute the ISR.
5) At the end of the ISR, it POPs the return address from the stack and loads it back into PC.
6) This is how the processor return to the very next instruction in the program.


Main Program
I S R
Interrupt
I/O device is ready for a data Transfer
Push PC → Stack
PC ← ISR address
Perform data Transfer with I/O device
Push PC ← Stack
(Get back the return address)

	INTERRUPT DRIVEN I/O	POLLING (PROGRAMMED I/O)
1	**I/O device interrupts the processor whenever it wants to perform a data Transfer.**	**Processor periodically checks (polls) the status of every I/O device to know if it wants to perform a data Transfer.**
2	**Processor is free** to carry on its own operations, hence **saves system time**.	**Processor is busy** in constantly checking all I/O devices, hence **system time wasted**.
3	**Additional hardware required** to handle interrupts. E.g.:: 8259 Programmable interrupt controller.	Additional hardware **not required**.
4	Increases **cost and complexity** of the system.	System is **cheaper and less complex**.
5	Interrupt **priority** has to be managed through software or through hardware.	No such issue.
6	Interrupt **vector addresses** (ISR Addresses) need to be stored in an Interrupt Vector table - **IVT**.	No such issue.

Types of interrupts

1. VECTORED AND NON VECTORED INTERRUPTS

A key element in interrupt handling is the ISR address.

If an interrupt has a fixed ISR address, it is called a Vectored interrupt.
Such interrupts are **executed faster** as the ISR address is known to the processor.
But such interrupts are **rigid**. Since they have a fixed ISR address they can serve only **one device**.
They cannot accept interrupts from multiple devices.
So they cannot expand the interrupt structure.
E.g.:: NMI interrupt of 8086 (Has a fixed vector number i.e. 2)

If an interrupt does not have a fixed ISR address, it is called a Non-Vectored interrupt.
Such interrupts are **executed slower**.
The ISR address is **obtained from the** interrupting **device**, usually an interrupt controller like **8259**.
But such interrupts are **flexible**.
Since they don't have a fixed ISR address they **can accept interrupts from multiple devices**.
So they can be used to **expand the interrupt structure**.
E.g.:: INTR interrupt of 8086 (Can service any vector number from 0… 255)

2. MASKABLE AND NON MASKABLE INTERRUPTS

Masking an interrupt means disabling it.

A **Maskable** interrupt is an interrupt that **can be disabled**.
If disabled, whenever this interrupt occurs, the processor will ignore it and simply continue the main program. Such interrupts are generally used to handle low priority, non-critical events like keyboard presses which can be easily disabled (Keypad can be locked)
E.g.:: INTR interrupt of 8086 (is disabled when Interrupt Flag is 0)

A **Non-Maskable** interrupt is an interrupt that **cannot be disabled**.
Whenever this interrupt occurs, the processor will have to service it.
Such interrupts are generally used to handle high priority, critical events like over-heating of the mother board, power failure etc.
E.g.:: NMI interrupt of 8086 (can never be disabled)

3. SOFTWARE AND HARDWARE INTERRUPTS

This is based on how the interrupt occurs.

If an interrupt is caused by **writing an instruction**, it is called a **software interrupt**.
Software interrupts are predictable events and are given by the programmer.
E.g.:: INT n instruction of 8086 (n can be anything between 0… 255)

If an interrupt is caused by **a signal on an external pin**, it is called a **hardware interrupt**.
Hardware interrupts are un-predictable events and are given by external devices.
E.g.:: NMI and INTR pins of 8086

Bharat
Acharya
Education

COA Viva Questions

Learn the full subject at:
www.BharatAcharyaEducation.com

6 Months | Unlimited Views | @999/-

Website	www.BharatAcharyaEducation.com
Mobile	+91 9820408217
Email	bharatsir@hotmail.com
Youtube	youtube.com/c/BharatAcharyaEducation
Instagram	@BharatAcharyaEducation
Facebook	facebook.com/BharatAcharyaEducation

Examiner

What are the three types of buses?

Question 1:

You

Address bus | Carries the address for the current memory or I/O operation.

Data bus | Transfers data between the processor and memory or I/O.

Control bus | Issues control signals to decide the type of operation like read, write etc.

Together they are called the SYSTEM bus

Bharat Acharya Education

Examiner

What is an instruction cycle?

Question 2:

You

The total process of fetching an instruction from the memory, decoding the opcode to understand the operation and finally executing the instruction is called an instruction cycle.

Bharat Acharya Education

Examiner

What is PC?

Question 3:

You

PC | Program Counter

It is a register present inside the processor. It contains address of the next instruction. PC gets incremented as soon as any instruction is fetched.

Bharat Acharya Education

Examiner

What is SP?

Question 4:

You

SP | Stack Pointer

It is a register present inside the processor. It contains address of the top of stack. SP gets decremented in every Push and incremented in every Pop operation.

Bharat Acharya Education

Examiner

What are GPRs?

Question 5:

You

GPRs | General Purpose Registers

These are registers present inside the processor. They are available to the programmer for storing operands and getting results in a program.

Bharat Acharya Education

Examiner

What is a compiler?

Question 6:

You

A compiler is a program that converts a higher language code (E.g.: C, C++) into machine language that can be executed by the processor.

Bharat Acharya Education

Examiner

What is the difference between Von Neumann and Harvard Model?

Question 7:

You

Von Neumann Model

Common memory used to store programs and data. E.g.:: 8085, 8086.

Harvard Model

Separate program memory and data memory. E.g.:: 8051.

Bharat Acharya Education

Examiner

Explain Flag register?

Question 8:

You

It gives the status of the current result. It has flag bits that indicate if there was a carry, if the result was zero, the sign of the result, its parity etc. Flags are changed by the ALU after every operation.

Bharat Acharya Education

Examiner

Why are signed numbers stored in 2's complement form?

Question 9:

You

Because it does not produce a negative zero. 2's complement of any number gives –(that number) but 2's complement of 0 gives back 0 itself.

Bharat Acharya Education

Examiner

Why did you choose engineering?
To mug up answers or to
understand the subject?

You

(Answer this one Yourself!)

Lets "Understand" the subject!
www.BharatAcharyaEducation.com

Computer Organisation & Architecture

List of Video topics

- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Internal Examiner

What is half adder and a full adder?

Question 10:

You

Half Adder | Adds the two current bits. Produces a sum and a carry. Cannot be scaled to a bigger adder.

Full Adder | Adds the two current bits along with the carry of the previous stage. Produces a sum and a carry. Can be scaled (combined) to form a bigger adder.

Refer Video: **COA | Adder Circuits**
www.BharatAcharyaEducation.com

Examiner

How is subtraction performed?

Question 11:

You

We never really “subtract”
Subtraction is performed by the same circuit used for addition by simply adding the 2’s complement of the number.

Draw and show the adder/Subtractor circuit if asked!

Refer Video: **COA | Adder Circuits**
www.BharatAcharyaEducation.com

Internal Examiner

Why cant we do multiplication by repeated addition?

Bonus Question

You

Because it is too slow!
1000 x 1000 will require 1000 steps!

Hence we use faster algorithms like Booth's Algorithm!

Refer Video: **COA | Booth's Algorithm**
www.BharatAcharyaEducation.com

Examiner

Briefly explain Booth's algorithm?

Question 12:

You

Keep checking bit transitions in the multiplier from right to left starting with an imaginary 0 beyond the LSB. Do the following actions:

0 to 1 | Subtract Multiplicand, Right Shift

1 to 0 | Add Multiplicand, Right Shift

0 to 0 | Right Shift

1 to 1 | Right Shift

Refer Video: **COA | Booth's Algorithm**

www.BharatAcharyaEducation.com

Internal Examiner

Why cant we do division by repeated subtractions?

Bonus Question

You

Because it is too slow!
1000 ÷ 1 will require 1000 steps!

Hence we use faster algorithms like restoring and non restoring division!

Refer Video: **COA | Division**
www.BharatAcharyaEducation.com

Examiner

Explain restoring division in brief?

Question 13:

You

At each step, **left shift** the dividend and **subtract** the divisor.

If result is positive (or 0) then successful. Quotient bit is 1. Restoration not needed.

If result is negative, unsuccessful. Quotient bit is 0. Restore by adding back the divisor.

Repeat for all bits of dividend.

Refer Video: **COA | Division**
www.BharatAcharyaEducation.com

Examiner

Explain non restoring division in brief?

Question 14:

You

At each step, **left shift** the dividend and **subtract** the divisor.

If result is positive (or 0) then successful. Quotient bit is 1. Restoration not needed.

If result is negative, unsuccessful. Quotient bit is 0. Don't Restore! Next step will be addition instead of subtraction.

Repeat for all bits of dividend.

Refer Video: **COA | Division**

www.BharatAcharyaEducation.com

Examiner

Describe Normalization and normalized form of a number ?

Question 15:

You

Normalization represents a number such that there must be only one non-zero digit to the left of the point.

Normalized form of a number

$(-1)^S \times 1.M \times 2^E$

S: Sign | M: Mantissa | E: Exponent

Refer Video: **COA | Floating Pt. Formats**
www.BharatAcharyaEducation.com

Examiner

Convert 0101.001 into Normalized form?

Question 16:

You

$(-1)^0 \times 1.0101001 \times 2^2$

Ref Video: **COA | Floating Pt. Formats**
www.BharatAcharyaEducation.com

Computer Organisation & Architecture

List of Video topics

- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Examiner

Explain Single Precision format?

Question 17:

You

IEEE754 32bit Single Precision

S	E	M
(1)	(8)	(23)

Bias value: 127

Ref Video: **COA | Floating Pt. Formats**
www.BharatAcharyaEducation.com

Examiner

Explain Double Precision format?

Question 18:

You

IEEE754 64bit Double Precision

S	E	M
(1)	(11)	(52)

Bias value: 1023

Ref Video: **COA | Floating Pt. Formats**
www.BharatAcharyaEducation.com

Examiner

Explain Floating point Addition in brief?

Question 19:

You

Compare exponents. If Equal, add the mantissas, else Shift the mantissa till exponent becomes equal, then add the mantissa.

Ref video: **COA | Floating Pt. Addition**
www.BharatAcharyaEducation.com

Examiner

What is Floating point overflow exception?

Question 20:

You

When the exponent is too large to be stored. It is denoted by exponent all 1's. Such a number is also called NaN.

Don't waste Your chance to learn such a wonderful subject by mugging up answers.

Learn the concepts at:
www.BharatAcharyaEducation.com

Examiner

What is Floating point overflow exception?

Question 21:

You

When the exponent is too small to be stored. It is denoted by exponent all 0's. Such a number is also called a DeNormal Number.

Don't waste Your chance to learn such a wonderful subject by mugging up answers.

Learn the concepts at:
www.BharatAcharyaEducation.com

Examiner

How do you represent 0 and 1 in Single precision format?

Question 22:

You

0:

S = 0 | E = 0 | M = 0

1:

S = 0 | E = $(127)_2$ | M = 0

Don't waste Your chance to learn such a wonderful subject by mugging up answers.

Learn the concepts at:

www.BharatAcharyaEducation.com

Examiner

What is the difference between a subroutine (procedure) and a macro?

Question 23:

You

When we CALL a subroutine, the program control shifts from the main program to the subroutine, and after execution, returns to the next instruction after CALL.
When we invoke a macro, the macro code is pasted into our program at the location where the macro was invoked.

Bharat Acharya Education

Examiner

What is an instruction format?

Question 24:

You

Label: Opcode, Operands; Comment

Label | Location Identifier (optional)
Opcode | Binary code of the operation (compulsory)
Operands | Values taking part in the operation (optional)
Comment | Purpose of the instruction (optional)

Ref Video: **COA | Instructions**
www.BharatAcharyaEducation.com

Examiner

What are the types of instructions?

Question 25:

You

Data Transfer
E.g.: MOV, LOAD, STORE etc.

Data Processing
E.g.: ADD, SUB, AND etc.

Control instructions
E.g.: JMP, CALL etc.

Ref Video: **COA | Instructions**
www.BharatAcharyaEducation.com

Examiner

What is instruction level pipelining?

Question 26:

You

Overlapping different stages of an instruction is called pipelining.
E.g.:: Fetch, Decode, Execute, Store etc.

Ref Video: **COA | Pipelining**
www.BharatAcharyaEducation.com

Internal Examiner

Give examples of pipelined processors?

Question 27:

You

8086 | 2 stage
Fetch, Exec.

80386 | 3 stage
Fetch, Decode, Exec.

Pentium | 5 stage
Fetch, Decode, Addr Gen, Exec, Store

Ref Video: **COA | Pipelining**
www.BharatAcharyaEducation.com

Computer Organisation & Architecture

List of Video topics

- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Examiner

What are Pipelining Hazards?

Question 28:

You

Data Hazard
Structural Hazard
Control Hazard

Ref Video: **COA | Pipelining**
www.BharatAcharyaEducation.com

Examiner

What do you mean by addressing modes. List some with examples?

Question 29:

You

It is the manner in which an operand is given in the instruction.
Most popular ones are…
Immediate | Register | Direct | Indirect | Implied

For examples:
Ref Video: **COA | Addressing Modes**
www.BharatAcharyaEducation.com

Examiner

What are the micro-operations required for fetching?

Question 30:

You

T1: MAR ← PC
T2: MBR ← instruction from memory
T3: IR ← MBR
PC ← PC + 1

Ref Video: **COA | Micro Operations**
www.BharatAcharyaEducation.com

Examiner

What is an instruction cycle, machine cycle and T-state?

Question 31:

You

Instruction cycle |Total process of fetching, decoding and executing an instruction.

Machine cycle | One complete operation of the system bus.

T-state | One clock cycle.

Instruction cycle has several machine cycles. Machine cycles have several T-states.

www.BharatAcharyaEducation.com

Examiner

Give characteristics of a RISC processor?

Question 32:

You

RISC | Reduced Instruction Set Computers

- Fixed sized instructions
- Mainly Register based operations
- Fewer addressing modes
- Smaller instruction set
- Simpler instructions
- Prefer Hardwired Control Units
- Suited for µControllers

Ref Video: **COA | Control Units**
www.BharatAcharyaEducation.com

Examiner

Give characteristics of a CISC processor?

Question 33:

You

CISC | Complex Instruction Set Computers

- Variable sized instructions
- Mainly Memory based operations
- More addressing modes
- Bigger instruction set
- Complex instructions
- Prefer μ-Programmed Control Units
- Suited for μProcessors

Ref Video: **COA | Control Units**
www.BharatAcharyaEducation.com

Examiner

What are the two main types of Control Units?

Question 34:

You

Hardwired Control Unit
Microprogrammed Control Unit

Ref Video: **COA | Control Units**
www.BharatAcharyaEducation.com

Examiner

What is a Hardwired Control Unit?

Question 35:

You

It generates control signals using dedicated HARDWARE.
Common types are
1) State Table method
2) Delay Element method
3) Sequence Counter method

Video: **COA | Hardwired Control Unit**
www.BharatAcharyaEducation.com

Examiner

What is a Microprogrammed Control Unit?

Question 36:

You

It generates control signals for any instruction using its μ instructions.

Typical modern Microprogrammed Control Units are made following Wilke's Design.

COA | Microprogrammed Control Unit

www.BharatAcharyaEducation.com

Examiner

What are μ Programs?

Question 37:

You

A set of μ instructions responsible for generating control signals for a particular instruction is called its μ program. μ programs are stored in the control memory.

This subject lays Your foundation for understanding deep machine learning concepts. Don't waste this opportunity by mugging up the answers.

Learn the concepts at:
www.BharatAcharyaEducation.com

Computer Organisation & Architecture

List of Video topics

- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Examiner

What are μ instruction sequencing techniques?

Question 38:

You

Two main techniques

1) Dual Address Field

2) Single Address Field

COA | Microprogrammed Control Unit

www.BharatAcharyaEducation.com

Examiner

What are μ instruction formats?

Question 39:

You

Two main formats

1) Horizontal μ instructions

2) Vertical μ instructions

COA | Microinstruction Formats

www.BharatAcharyaEducation.com

Examiner

Compare Hardwired Control Unit and Microprogrammed Control Unit?

Question 40:

You

Hardwired Control Unit

Faster | No Control Memory needed | Suited for RISC

Hardwired Control Unit

Flexible | Supports Emulation | Control Memory needed | Suited for CISC

COA | Microprogrammed Control Unit

www.BharatAcharyaEducation.com

Examiner

List “typical” sizes of these memories?

Question 41:

You

Hard Disk | **1 TB**
CD | **700 MB**
DVD | **4.7 GB**
Pen Drives | **2 GB – 128 GB**
RAM | **4GB, 8GB, 16GB, 32GB**
Cache | **1 MB – 4 MB**

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

Compare SRAM and DRAM?

Question 42:

You

Static RAM

Faster | Uses Flip-Flops | Expensive | used in Cache

Dynamic RAM

Slower | Uses Capacitors | Cheaper | used in main memory

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

What are advanced DRAMs?

Question 43:

You

RDRAM | Rambus DRAM
SDRAM | Synchronous DRAM

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

What are the types of ROM?

Question 44:

You

ROM | Read Only Memory
PROM | Programmable ROM
EPROM | Erasable PROM
EEPROM | Electrically EPROM
Flash ROM | Electrically EPROM Most Popular in modern systems

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

Where does your phone store images, songs etc?

Question 45:

You

In secondary storage implemented using Flash ROM

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

What does primary memory consist of?

Question 46:

You

RAM and ROM.
RAM for operations
ROM mainly fro the BIOS

Ref video: **COA | Memory Hierarchy**
www.BharatAcharyaEducation.com

Examiner

List important Page replacement policies?

Question 47:

You

FIFO, LRU, LFU, OPT

For solved examples on each of them, refer the video:

COA | Numericals on Page Replacement
www.BharatAcharyaEducation.com

Examiner

What are dirty pages ;-)

Question 48:

You

These are pages modified in the main memory.

Ref video: **COA | Paging**
www.BharatAcharyaEducation.com

Computer Organisation & Architecture

List of Video topics

- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Examiner

What is TLB?

Question 49:

You

TLB | Translation Look-aside Buffer
Contains the most recently used entries of the page table to make page translation faster.

Ref video: **COA | Paging**
www.BharatAcharyaEducation.com

Examiner

What is a page fault?

Question 50:

You

Page Fault

When the desired page is not present in the main memory it is called a miss or a page fault.

Ref video: **COA | Paging**
www.BharatAcharyaEducation.com

Examiner

List some memory characteristics?

Question 51:

You

Reliability, Storage Capacity, Hit Ratio, Avg. Access Time, Avg. Cost/bit, location, Access mode, Access type etc…

Ref Video: **COA | Memory Characteristics**
www.BharatAcharyaEducation.com

Examiner

What are interleaving tecniques?

Question 52:

You

Lower order interleaving and higher order interleaving.

Ref Video: **COA | Memory Interleaving**
www.BharatAcharyaEducation.com

2 options: Mug up all this and bore yourself
Or, learn the subject the right way and
Enjoy its Vast concepts!

Examiner

What is cache consistency?

Question 53:

You

When data in cache memory and main memory has same value, the cache is consistent. Main policies: Write Through and Write back.

Ref Video: **COA | Cache memory**
www.BharatAcharyaEducation.com

Examiner

What are cache layouts?

Question 54:

You

Look Through Cache and Look Aside Cache.

Ref Video: **COA | Cache Memory**
www.BharatAcharyaEducation.com

Examiner

What is MESI protocol?

Question 55:

You

Modified, Exclusive, Shared, Invalid.

Video Ref: **COA | Cache Memory**
www.BharatAcharyaEducation.com

Examiner

What are cache mapping techniques?

Question 56:

You

Associative Mapping
Direct Mapping
Set Associative Mapping

This is one of the BEST topics you learn in this subject. My favorite amongst all!

COA | Cache Mapping Techniques
www.BharatAcharyaEducation.com

Examiner

What are the main types of multi-processor systems?

Question 57:

You

Closely Coupled and Loosely Coupled systems.

www.BharatAcharyaEducation.com

Examiner

What are Bus Arbitration techniques?

Question 58:

You

Daisy Chaining, Polling and Independent requests.

Ref Video: **COA | Bus Arbitration**
www.BharatAcharyaEducation.com

Examiner

What are I/O data transfer techniques?

Question 59:

You

Polling/ Programmed I/O
Interrupt driven I/O
DMA
I/O Processors

Ref Video: **COA | I/O Data Transfers**
www.BharatAcharyaEducation.com

Examiner

What are I/O mapping techniques?

Question 60:

You

Memory mapped I/O and I/O mapped I/O (also called Isolated I/O)

Ref Video: **COA | I/O Mapped I/O**
www.BharatAcharyaEducation.com

Examiner

Give characteristics of interrupts?

Question 61:

You

Software, hardware, vectored, non-vectored, edge triggered, level triggered, Maskable, non Maskable.

Ref Video: **COA | Interrupt Driven I/O**
www.BharatAcharyaEducation.com

Examiner

What are some popular DMA transfer techniques?

Question 62:

You

Block Transfer (Burst Mode)
Cycle Stealing (Single Byte Transfers)
Demand Transfer
Hidden Mode (Transparent Mode)

Ref Video: **COA | Direct Memory Access**
www.BharatAcharyaEducation.com

Examiner

Name a popular I/O Processor?

Question 63:

You

8089 I/O Processor is used with 8086 μ P.

Ref Video: **COA | I/O Processor**
www.BharatAcharyaEducation.com

Examiner

Give Flynn's Classification?

Question 64:

You

It classifies multiprocessor systems into 4 different categories, based on the instruction stream and the data streams.

1) **SISD** | Single Instruction Single Data
2) **SIMD** | Single Instruction Multiple Data
3) **MISD** | Multiple Instruction Single Data
4) **MIMD** | Multiple Instruction Multiple Data

www.BharatAcharyaEducation.com

Examiner

What is serial and parallel communication?

Question 65:

You

Serial Communication

One bit at a time | Slower | Cheaper | Long Distance

Parallel Communication

Multiple bits at a time | Faster | Costlier | Short Distance

www.BharatAcharyaEducation.com

www.BharatAcharyaEducation.com

Video Courses for You

- Microprocessor | 8085
- Microprocessor | 8086
- Microprocessor | 80386
- Microprocessor | Pentium
- Microcontroller | 8051
- Microcontroller | ARM
- Computer Organisation & Architecture

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Bharat Acharya Education

Computer Organization
& Architecture
VIVA | 2019

Computer Organisation & Architecture

List of Video topics

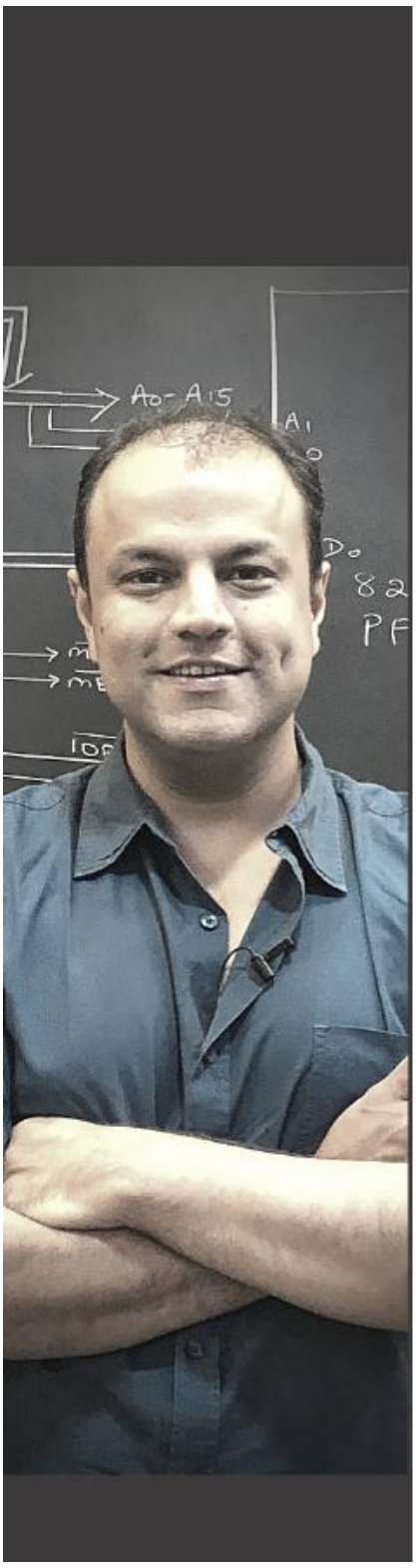
- Introduction
- Von Neumann Model
- Number representation
- Adder Subtractor circuits
- Multiplication including Booth's Algorithm
- Restoring Division
- Non Restoring Division
- Floating Point Number Representation
- Floating Point Conversion Numericals
- Floating Point Addition Algorithm
- Instruction Formats and Types
- Instruction cycle state transition
- Addressing Modes with Examples of 8085/ 8086 etc.
- Pipelining
- Micro-Operations
- Hardwired Control Units
- Microprogrammed Control Units
- Micro-Instruction Sequencing
- Horizontal v/s Vertical Micro Instruction Formats
- Nano-Programming
- Memory Hierarchy
- Memory characteristics
- Virtual Memory Management - Paging & TLB
- Page Replacement Theory & Numericals
- Cache Architectures
- Cache Mapping Techniques
- Bus Arbitration
- Memory Mapped I/O v/s I/O Mapped I/O
- Interrupt Driven I/O
- DMA based I/O
- Flynn's Classification
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Bharat Acharya Education

Computer Organization & Architecture VIVA | 2019



Microprocessor 8085

List of Video topics

- Introduction
- Architecture
- Flag Register
- Addressing Modes
- Instruction Set (Every Instruction with examples)
- Assembly Programming (Several Videos)
- Timing Diagrams of Machine cycles
- Timing Diagrams of Instructions
- Stacks & Subroutines
- Delays & Delay Routines
- Interrupt Structure
- SIM and RIM Instructions with Sq. Wave Program
- 8259 PIC (Theory, Programs, Interfacing)
- 8255 PPI (Theory, Programs, Interfacing)
- 8254 PIT (Theory, Programs, Interfacing)
- 8237 DMAC (Theory, Programs, Interfacing)
- Full System Memory & I/O Designing
- Memory Mapped I/O and I/O Mapped I/O
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

Bharat Acharya Education

Computer Organization & Architecture VIVA | 2019

Microprocessor 8086

List of Video topics

- Introduction
- Pipelining
- Memory Segmentation
- Memory Banking
- Architecture
- Flag Register
- Addressing Modes
- Instruction Set (Every Instruction with examples)
- Assembler directives
- Assembly Programming (Several Videos)
- DOS Interrupt - INT 21H
- Interrupt structure with IVT
- Minimum Mode with Timing Diagrams
- Maximum Mode with Timing Diagrams
- 8086-8087 Interfacing
- 8259 PIC (Theory, Programs, Interfacing)
- 8255 PPI (Theory, Programs, Interfacing)
- 8254 PIT (Theory, Programs, Interfacing)
- 8237 DMAC (Theory, Programs, Interfacing)
- Full System Memory & I/O Designing
- Many more...

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com

www.BharatAcharyaEducation.com

Registration Process

- Log into our website: www.BharatAcharyaEducation.com
- Select your desired course
- Make your payment
- Start Learning!

Payment options

Our website accepts the following payment methods:

- Netbanking
- Debit/ Credit Cards
- Wallets like Paytm etc

For Any further Payment assistance
OR
For International Students
Please call/ Whatsapp us on:

+91 9820408217

6 Months | Unlimited Views | 999/-
GST applicable @18% as per Govt. of India

www.BharatAcharyaEducation.com

www.BharatAcharyaEducation.com

Additional Benefits

- Free PDF with almost every video containing full theory answers and circuit diagrams
- Certificate at the end of the course
- Doubt solving via WhatsApp on +919829498217 subject to availability of the faculty

Rules and regulations

- All videos are copyright protected.
- UNAUTHORISED DOWNLOADING OR CIRCULATION of any content on our website or mobile app are subject to LEGAL ACTION as per INDIAN COPYRIGHT ACT.
- One Login Id and the subsequent subscription is for one individual and can be used on upto 4 devices, though not simultaneously.
- GST applicable as per Govt. rules.

Contact Us

Address: 1, Vagholkar Apts, RS Road, Thane (W) -400601
E: bharatacharyaeducation@gmail.com
M: +91 9820408217

6 Months | Unlimited Views | 999/-

www.BharatAcharyaEducation.com